Universidad Nacional de Córdoba

Facultad de Matemática, Astronomía, Física y Computación

Grupo Teoría de la Materia Condensada

Modelado del proceso de reconocimiento de orientaciones en imágenes usando redes neuronales

Trabajo Final de la Licenciatura en Física
escrito por Camila Britch

Director:
Dr. Francisco Antonio Tamarit

Agradecimientos

- A mi papas, Javier y Clau. Que pese a sus advertencias sobre la facultad, me bancaron en mi decisión de entrar a la carrera y me apoyan en incondicionalmente en la vida.
- A mis hermanas, Jose y Agus, que me alegran absolutamente todos los días.
- Al Ari, el Pela, quien me acompaña hace más de nueve años y con quien sé que puedo contar para absolutamente cualquier cosa.
- A Pancho, gran docente y persona que me acompañó en este trayecto final para poder hacer la tesis.
- A Juan Porta, que sin tener responsabilidades asumidas se bancó todas mis preguntas y sin el cual no hubiese podido ni empezar este trabajo. Un millón de gracias.
- A Pablito, quién además de aliento constante, cuando me quede sin herramienta de trabajo, me presto -sin dudarlo un segundo- su compu.
- A todes mis compañeres de carrera y facultad, que fueron montones en estos MUCHOS años. Compartiendo alegrías, sufrimientos y sobretodo mates en sala de estudio.
- En particular al Joaquito, que este año, lleno de cambios, fue mi mayor pilar emocional.
- A la educación pública, laica, gratuita y de calidad de FaMAF.

Resumen

En este trabajo se aborda el problema de diseñar diferentes redes neuronales con aprendizaje supervisado para el reconocimiento de ángulos de rotación en imágenes. Dicho trabajo se aborda no solo por su valor dentro de la inteligencia artificial, sino también por su gran relación con el aspecto biológico del cerebro de mamíferos, en particular del área de reconocimiento de imágenes visuales en el córtex visual o sector V1 y la correspondiente formación de patrones de reconocimiento de orientaciones en ella, llamados "mapas de preferencia orientacional". En particular, se trabaja con tres modelos diferentes de redes neuronales convolucionales: de clasificación, regresión y de reconstrucción de la imagen de entrada. Si bien la tarea de reconocer ángulos de rotación para su posterior utilización para enderezar la imagen no es novedoso, en este trabajo se diseña una red Auto-encoder para unificar estos dos pasos en una única red. Es decir, la tarea específica de dicha red es extraer las características esenciales tanto de la imagen como de su rotación para ser capaz de reconstruirla de forma enderezada a la salida. Dentro de cada modelo se exploran diferentes arquitecturas, correspondientes a cada tarea, obteniendo en todas un muy buen desempeño.

Resumen en inglés

This thesis deals with the problem of designing different neural networks with supervised learning for the recognition of angles of rotation in images. This topic is addressed not only for its importance within artificial intelligence, but also for its great relationship with the biological aspect of the mammalian brain, particularly the visual image recognition area, the "V1" visual cortex, and pattern formation recognition of orientations in it, called "maps of orientational preference". In particular, we work with three different models of convolutional neural networks: classification, regression and reconstruction of the input image. Although the task of recognizing rotation angles for their later use to straighten the image is not new, in this work an Auto-encoder network is designed to unify these two steps in a single network. That is, the specific task of said network is to extract the essential characteristics of both the image and its rotation in order to be able to reconstruct it in a straightened way at the output. Within each model, different architectures are explored, corresponding to each task, obtaining great performance in all of them.

Índice general

Introducción

Para poder hablar de Inteligencia Artificial (IA) lo primero que deberíamos poder hacer es dar una definición de la misma. La versión académica tradicional nos dice que, en el campo de la informática, la IA es la "disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico". Dependiendo del autor, hay diferentes enfoques de la definición que se centran en los *sistemas* que actúan, mientras que otros prefieren focalizar en en *desarrollo de sistemas* capaces de "pensar". En este trabajo destacamos dos definiciones, la dada por Keith Downing: *"hacer lo correcto en el momento adecuado, desde el punto de vista de un observador humano externo"* (Downing (2015)), y la de Elaine Rich: *"el estudio de cómo hacer que las computadoras hagan cosas que, por ahora, los humanos hacemos mejor"* (Rich (1985)). Ambas se centran más en el resultado del proceso que en el proceso en sí, dando lugar a la llamada definición de *caja negra* y dejando clara la atención especial en problemas que los seres humanos resolvemos **mejor** que las máquinas. Dada esta definición móvil, muchos sistemas a los que nos hemos acostumbrado se considerarían, hace unas décadas, ejemplos de IA y problemas que hoy consideramos que requieren IA, pues son tareas difíciles para una computadora, en el futuro nos parecerá normal que las máquinas sean capaces de realizarlas. Por ejemplo, los primeros desarrollos de IA se centraron en seguir reglas lógicas, lo que les permitió realizar tareas como demostrar teoremas matemáticos o resolver rompecabezas, mientras que otros problemas, como construir sistemas de reconocimiento de voz o identificación de objetos en imágenes, sólo consiguieron resultados comparables a los obtenidos por un ser humano con la llegada de las técnicas de aprendizaje

automático y el desarrollo del *aprendizaje profundo* (deep learning). Cuando hablamos específicamente del aprendizaje automático se entiende como un proceso por el cual una máquina es capaz de mejorar su habilidad en la resolución de un problema mediante de la adquisición de conocimiento basado en experiencias previas.

Como sucede con muchas disciplinas, se desarrollaron diferentes perspectivas para abordar el problema del aprendizaje automático. Pedro Domingos (Domingos (2015)) distingue cinco familias diferentes.

- Los *simbólicos* centran su atención en la interpretación filosófica, lógica y psicológica del aprendizaje: un proceso de inducción.
- Los *analógicos* basan el aprendizaje en la extrapolación a partir de ejemplos conocidos, mediante la realización de juicios de similitud. Su razonamiento por analogía tiene bases psicológicas y, en ocasiones, se acaba traduciendo en un problema matemático de optimización.
- Los *bayesianos* utilizan técnicas estadísticas de inferencia probabilística, basadas en última instancia en el teorema de Bayes, como sucede con las redes bayesianas.
- Los *evolutivos* extraen sus conclusiones de la teoría de la evolución de Darwin y de las bases de la genética, de donde se inspiran para desarrollar técnicas como los algoritmos genéticos.
- Los *conexionistas* se inspiran en el cerebro humano, intentando hacer ingeniería inversa de su funcionamiento con la ayuda de físicos y neurocientíficos (aunque sus modelos no siempre acaben siendo biológicamente plausibles).

En este trabajo nos focalizaremos en la teoría conexionista, que es la familia de los investigadores que emplean *Redes Neuronales Artificiales* (RNA) para la resolución de problemas. Con su gran historia, con altibajos desde sus orígenes, las RNA han sido capaces de alcanzar una eficacia comparable con la del hombre en distintos problemas complejos de reconocimiento de patrones.

La idea principal en las que se basan las RNA es intentar encontrar la solución a los problemas combinando las contribuciones realizadas por un gran número de unidades de procesamiento simples que se encuentran conectados entre sí. Dichas unidades se denominan *neuronas* y las conexiones o sinapsis representan el "conocimiento adquirido". Las RNA se inspiran en el funcionamiento del cerebro humano y mientras en ocasiones intentan modelar fielmente alguno de los procesos que realiza nuestro cerebro, otras veces hacen una interpretación más libre de cómo se podría resolver un problema usando un sistema distribuido con múltiples unidades de procesamiento, aunque esa forma de resolver el problema no resulte plausible desde el punto de vista biológico. Las interpretaciones más cercanas a las ciencias biológicas basan su trabajo en los descubrimientos que la neurociencia nos ha proporcionado en su intento de hacer ingeniería inversa del funcionamiento del cerebro humano. Se intentan mantener fieles a la biofísica de las neuronas biológicas.

Por su gran capacidad de procesamiento, destacan hoy, de entre las múltiples arquitecturas neuronales profundas, en la tarea particular de análisis de imágenes, las *Redes Neuronales Convolucionales* (CNN). Estas son un tipo de RNA con aprendizaje supervisado que procesa sus capas imitando al cortex visual del ojo humano para identificar distintas características en las entradas que permite procesar escenas visuales preservando la topología del estimulo. Para ello, una CNN contiene varias capas ocultas especializadas y con conjuntos de filtros convolucionales de una o más dimensiones. Un tipo particular y muy exitoso de CNN son los *Auto-encoder Convolucionales* (AEC) que permiten aprender, a través de una representación de baja dimensionalidad de los datos (espacio latente) a procesar la información disponible, lo cual evita el paso de clasificación de los estímulos y permite modelar con más realismo la forma automática en que los humanos y otras especies animales procesamos escenas visuales.

En este trabajo, si bien nos interesa el problema de aprendizaje automático, buscamos entender como se comportan los cerebros humanos, siguiendo la línea de trabajo que se plantean en los trabajos realizados por Yudy Carolina Daza Caro (Daza Caro (2017), Gleiser et al. (2018)) y Carolina Beatriz Tauro (Tauro (2012), Tauro (2012),Tauro et al. (2014)) en los que se estudia la forma de procesar in-

formación visual en el cerebro de mamíferos, en particular en el área de la corteza visual, también llamada V1, la cual presenta respuestas a estímulos visuales de barras y bordes con determinada orientación que forman patrones espacio-temporales de actividad sincronizada llamados *"mapas de preferencia orientacional"*, para poder modelarla lo más fielmente posible a través de redes neuronales. A diferencia de los trabajos anteriores, la idea de esta tesina es diseñar redes neuronales convolucionales. Esta elección se funda en que son arquitecturas especializadas en la tarea de procesamiento de señales, particularmente el problemas de visión artificial.

El objetivo es diseñar diferentes arquitecturas de redes neuronales convolucionales para aprender a reconocer ángulos de rotación de la entrada dada, que en esta oportunidad será una imagen. En particular se usarán tres arquitecturas de conexiones sinápticas y umbrales de activación diferentes: la de clasificación, la de regresión y la reconstrucción de la imagen de entrada (AEC). Lo que se busca es que, dado un conjunto de datos, en este caso representado por una imagen previamente rotada, cada elemento quede representado por unos pocos parámetros que permitan, a las diferentes redes, dependiendo de su tarea, ser capaces de identificar, ya sea por clasificación, regresión o reconstrucción, el ángulo de rotación que posee la imagen de entrada.

Capítulo 1

Marco Teórico

1.1. Aprendizaje automático

El aprendizaje automático es lo que permite que una computadora sea creativa. En vez de funcionar a través de un programa capaz de resolver un problema dado, que especifique que es lo que la máquina tiene que hacer paso a paso, funciona a través de mecanismos o programas genéricos capaces de aprender de su propia experiencia, lo cual le permite, como hacen lo s seres vivos, adaptarse a nuevas experiencias, que en este caso vienen dadas en forma de datos recolectados previamente. En cierto modo, los algoritmos de aprendizaje automático son algoritmos que construyen otros algoritmos, de forma que la computadora se programa a sí misma a partir de bases de datos que recopilan experiencias previas sin que nosotros tengamos que programarla.

¿En qué situaciones es especialmente recomendable el uso de técnicas de aprendizaje automático? Básicamente, en tareas que nos gustaría poder automatizar pero que no hemos sido capaces de hacerlo (hasta ahora) por la complejidad que requeriría la programación de algoritmos tradicionales para resolver los problemas asociados a ellas. No se trata de problemas no resolubles, sino de problemas que las personas resuelven con relativa facilidad pero que son extremadamente difíciles de resolver utilizando un algoritmo secuencial diseñado específicamente para esa tarea. Los problemas de reconocimiento de patrones son un ejemplo representativo del tipo de problemas para los que el aprendizaje automático resulta especialmente

indicado. Los sistemas de identificación de objetos en imágenes requieren tener en cuenta múltiples factores variables: solapamientos, oclusiones, sombras, condiciones de iluminación y cambios de perspectiva. Esas variables dan lugar a un número tan grande de situaciones posibles, que resultaría imposible especificarlas de antemano. Sin embargo, si se dispone de datos suficientes, las técnicas de aprendizaje automático pueden ser de gran utilidad. En vez de diseñar un algoritmo a medida para resolver el problema, que siempre será frágil (además de muy costoso), se diseña un algoritmo que aprenda de los datos disponibles y cree el "programa" necesario para resolver el problema. El "programa" generado no tiene por que parecerse a un programa convencional, como los que usamos en computación científica. Por ejemplo, una red neuronal puede contener millones de parámetros ajustados numéricamente durante su proceso de entrenamiento y resultar completamente ininteligible para un ser humano. Esto es muy diferente a lo que sucede con un programa escrito manualmente en un lenguaje de programación de alto nivel, como resultado del uso de prácticas recomendadas y de la aplicación de sólidos principios de diseño.

Desde una perspectiva más académica, se denomina inteligencia computacional a la capacidad de aprendizaje de una computadora a partir de datos experimentales u observaciones. Mediante este término se engloban diversas técnicas a las que habitualmente llamamos *computación blanda* (soft computing), redes neuronales artificiales, computación evolutiva, aprendizaje estadístico, modelos gráficos probabilísticos o lógica difusa [fuzzy logic], entre otras. Las técnicas de computación blanda están diseñadas para tratar de forma explícita la presencia de imprecisión en los datos, incertidumbre e información incompleta. Como en la práctica esto es algo que sucede habitualmente, estas técnicas suelen ser especialmente robustas como técnicas de aprendizaje automático.

Si el algoritmo de aprendizaje tiene éxito, el programa creado automáticamente a partir de los ejemplos de su conjunto de entrenamiento, funcionará bien para nuevos ejemplos, nunca antes presentados al programa. Para ello debemos contar con un gran número de ejemplos. Cuanto más datos tengamos, más fácil resulta diferenciar patrones válidos de patrones irregulares o erróneos. Afortunadamente hoy resulta relativamente fácil y barato recopilar grandes cantidades de datos orientados al

problema a resolver pues es sencillo digitalizar y almacenar información.

En general, más allá de las redes neuronales, pero incluyéndolas, podemos decir que una primera clasificación de las técnicas de aprendizaje automático puede realizarse atendiendo a la filosofía utilizada en el proceso de adquisición del conocimiento, dentro del cual existen tres tipos de algoritmos de aprendizaje Berzal (2018).

- Aprendizaje supervisado. En este caso se dispone de un conjunto de datos de entrada previamente etiquetados (o sea, para el cual disponemos de la salida correcta) que permiten encontrar el conjunto de las sinapsis o parámetros que resuelven el problema para los ejemplos etiquetados, a los que llamaremos conjunto de entrenamiento.

- Aprendizaje no supervisado. En este caso no se cuenta con datos previamente etiquetados y el algoritmo debe procesar la información por si solo. Suele usarse para clasificar en pocas categorías. Será el método mismo el que decida cómo han de agruparse los datos del conjunto de entrenamiento (clustering) o qué tipo de patrones son más interesantes dentro del conjunto de entrenamiento (en las técnicas de extracción de reglas de asociación).

- Aprendizaje por refuerzo. En este caso el algoritmo funciona como un "agente autónomo" que explora un espacio desconocido y decide que acciones realizar mediante prueba y error. La información que recibe no son etiquetas sobre los datos de entrada sino si se ha equivocado o no, y dependiendo de esto recibirá premios y castigos. Así, crea la mejor estrategia posible para obtener la mayor recompensa. Esta estrategia definirá acciones a tomar ante cada situación que se enfrente.

A estos paradigmas hay que agregar enfoques híbridos y de ensamble que ganan día a día interés en la comunidad científica. En el amplio campo del aprendizaje automático, el enfoque general es predecir un resultado utilizando los datos disponibles. La tarea de predicción también se denomina "problema de clasificación" cuando el resultado representa diferentes clases y se denomina "problema de regresión" cuando el resultado es una medida numérica. En cuanto a la clasificación, la

configuración más común involucra solo dos clases, aunque puede haber más de dos. Este último caso se denomina "clasificación multiclase".

Es importante aclarar, antes de adentrarnos en este trabajo, que el mismo se encuadra dentro del paradigma de "aprendizaje supervisado". Se usarán diferentes modelos de redes neuronales con la intención de clasificar y de hacer regresiones. Dado que el problema a tratar involucra el procesamiento de imágenes, usaremos un tipo especial de redes neuronales llamadas *redes convolucionales*, las cuales serán oportunamente presentadas y descriptas. En este capítulo nos limitaremos a introducir en forma muy general el concepto de *redes neuronales artificiales* (RNA) como así también el problema que deseamos abordar con ellas. Es importante destacar que las redes neuronales artificiales no son los únicos modelos capaces de aprender de la experiencia la resolución de problemas complejos, pero sin duda hoy muestran un desempeño muy bueno para una gran diversidad de desafíos, ya sea en el campo del procesamiento de imágenes (área en la cual se encuadra el presente trabajo), el procesamiento de lenguaje natural y el análisis de series temporales, entre muchos otros problemas.

1.2. Redes Neuronales Artificiales

Las redes neuronales artificiales proporcionan un mecanismo mediante el cual se puede conseguir que una computadora aprenda. Se inspiran en el cerebro humano y construyen modelos informáticos formados por múltiples unidades relativamente simples, denominadas neuronas. Estas neuronas, o elementos de procesamiento, se conectan entre sí a través de una arquitectura de conexiones llamadas *sinapsis*. Mediante la manipulación de las conexiones entre las neuronas de la red, que se organizan en múltiples capas de neuronas (como el córtex cerebral, el cerebelo o la retina humana) se consigue que la red neuronal tenga el comportamiento deseado.

Las redes neuronales artificiales forman parte del conjunto de técnicas de aprendizaje automático, las cuales a su vez son un subconjunto de las técnicas utilizadas en Inteligencia Artificial. Estas redes pueden descubrir características a partir de los datos y también son capaces de crear nuevas características. De esa forma, no

sólo identifican características sino que aumentan el nivel de abstracción mediante la creación de jerarquías de niveles de características, en las que nuevas características se descubren de forma automática a partir de las características del nivel anterior.

Para entender entonces el funcionamiento de las RNA se debe estar familiarizado con algunos conceptos fundamentales de la neurobiología, el análisis matemático, el álgebra lineal, la teoría de probabilidad y la estadística. En particular, la física estadística guía muchos de los desarrollos teóricos, promoviendo la búsqueda de explicaciones neuronales para comportamientos mentales. Finalmente es necesario saber que para poder implementar las redes neuronales es necesario saber programar en algún lenguaje de alto nivel.

1.2.1. La neurona artificial

Las neuronas artificiales son unidades de cálculo que pretenden imitar el funcionamiento de las neuronas biológicas que constituyen los sistemas nerviosos de los animales. En particular, nos interesa poder conformar cerebros artificiales como conglomerados de neuronas artificiales. Estas neuronas son las unidades esenciales con las cuales se conforman las RNA. Desde el punto de vista computacional, lo único importante es que una señal proveniente de una sinapsis (una entrada) afecte al estado de una neurona que recibe el estímulo. Amplificada o atenuada de diferentes formas, se combina con otras señales de entrada para determinar el nivel de activación de la neurona. En función de ese nivel de activación, la salida de la neurona podrá variar (o no). De ahí que, habitualmente se modelen las sinapsis como un simple numero real, conocido normalmente como *peso* sináptico, asociado a la conexión. Aunque esa simplificación del funcionamiento de una neurona pueda parecer extrema, no lo es si capta los detalles realmente relevantes necesarios para explicar el funcionamiento del cerebro: la transmisión de señales de un punto a otro y la plasticidad de las conexiones neuronales, que se pueden modelar variando los pesos asociados a ellas. El modelo resultante no será siempre fiel a la biología, pero podrá captar la esencia del fenómeno y nos permitirá emular el proceso como un todo.

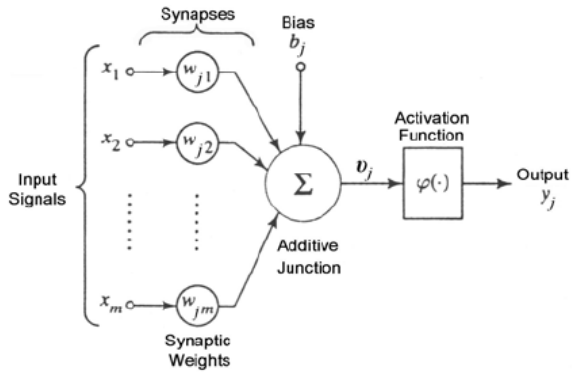


Figura 1.1: Modelo esquemático de la neurona artificial introducida por McCulloch y Pitts (McCulloch and Pitts (1943)) y generalizable a otras funciones de activación posibles.

El estímulo que llega a la neurona se puede pensar como la suma ponderada de los estímulos (entradas) que a ella le llegan ponderadas por el valor de las respectivas conexiones sinápticas. Este resultado será procesado por una función de activación que caracteriza a la dinámica de dicha neurona, como se ilustra en la Figura 1.1. El modelado matemático que lo ilustra se muestra a continuación:

$$y_j = \varphi(f(\vec{x}, \mu_j)), \tag{1.2.1}$$

donde $\vec{x} = (x_1, x_2, \cdots, x_m)$ es la entrada de la neurona, la función $\varphi(x)$ es la antes mencionada *función de activación*, la cual determina cual es la salida de la neurona dependiendo del estimulo neto recibido y finalmente μ_i es el umbral de activación que modela la diferencia de potencial de acción a partir del cual la neurona dispara una señal. Notemos que usamos el sub-índice i para hacer referencia a que nuestra función de activación modela la dinámica de la i-ésima neurona de un conjunto de muchas neuronas artificiales. Los estímulos que llegan a la neurona a través de cada una de sus N dendritas son sumados y ponderados por el valor de las N sinapsis que la conectan con las N neuronas pre-sinápticas:

$$f(\vec{x} - \mu_i) = \sum_{i=1}^{N} w_i x_i - \mu_i.$$

Una forma simple de tratar el umbral es pensar que se tiene una entrada extra, a la que se suele denominar x_0 la cual está fija en el valor -1. De esta forma, si asociamos al umbral con una sinapsis $w_0 = \mu_i$ tenemos que:

$$f(\vec{x} - \mu_i) = \sum_{i=0}^{N} w_i x_i.$$

Este modelo fue introducido en el año 1943 por Warren McCulloch y Walter Pitts McCulloch and Pitts (1943) en el contexto de procesamiento lógico de señales, motivo por el cual supusieron una función de activación binaria, con $y_i \in 0, 1$. Para ello introdujeron la llamada función Heaviside que tiene la siguiente forma:

$$H(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases}$$

Es interesante mencionar que, a pesar de que han pasado poco menos de ocho décadas desde entonces, la idea de usar unidades de umbral continúa dominando la modelización de neuronas artificiales para construir redes neuronales para aprendizaje automático, incluso en las arquitecturas más modernas. Sin embargo, ya no se usa, al menos no exclusivamente, la función Heaviside H y aparecen formas más adecuadas.

1.2.2. Funciones de activación

En lineas generales, las funciones de activación se pueden clasificar en dos grupos: por un lado, aquellas que son discretas, es decir, que su salida solo puede tomar un conjunto finito de valores (usualmente binarias o bipolares), y por otro, las llamadas funciones de activación continuas, en cuyo caso la salida puede tomar cualquier valor dentro de un intervalo (generalmente limitado como $[0, 1]$ o $[-1, 1]$).

Dentro de las funciones de activación continuas, se pueden utilizar diferentes familias de funciones no lineales, dependiendo de la complejidad de la red y del tipo de datos de salida con los cuales se trabaja. Es importante mencionar que nos interesan solo las funciones no lineales, pues si bien las lineales son mucho más fáciles

de manipular matemáticamente, no permiten abordar problemas interesantes de aprendizaje automático. Cuando pensamos en funciones continuas estamos haciendo un paralelismo entre el problema de aprendizaje y el problema de ajustar un familia de funciones con un conjunto de datos medidos. O sea, podemos pensar a estas neuronas como regresores universales.

La primera función no lineal utilizada, como ya hemos mencionado, fue la *función escalón, umbral o de Heaviside*. Pero no es el único modelo de función binaria: también podemos trabajar con la *función signo*, la cuales simétrica respecto del origen:

$$y = f_{\text{sgn}}(x) = \text{sgn}(x) = \begin{cases} +1 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}$$

Observamos que las funciones Heaviside y *signo* presentan una discontinuidad en el origen, por lo que se reemplazan por funciones continuas o continuas a trechos. Una de las más sencillas es la *función lineal con saturación* tanto en su versión binaria o bipolar simétrica. Podemos definirlas como

$$y = f_{\text{sls}}(x) = max(d, min(1, x))$$

donde d es 0 en el caso binario y -1 en el caso bipolar. A ésta última, Ronan Collobert la denominó como *tangente hiperbólica estricta* por su similitud con la tangente hiperbólica, aunque su derivada no este definida para $x = \pm 1$.

Veremos que el uso de funciones continuas y diferenciables facilitará los algoritmos de aprendizaje. La *funciones sigmoidales* satisfacen dichos requisitos por lo que son muy útiles en redes que se entrenan con el algoritmo de back-propagation, el cual se presentará en breve. Dos funciones sigmoidales muy populares en el aprendizaje automático son: la *función logística* y la función *tangente hiperbólica*. La primera tiene como imagen el intervalo $(0, 1)$. Su expresión matemática es

$$y = f_{\text{logistic}}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

La función *tangente hiperbólica*, en cambio, tiene imagen en el intervalo $(-1, 1)$ y se

define como:

$$y = f_{\tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Hoy en día soy muy importantes las *funciones de activación lineal rectificadas*, que son funciones no lineales que no requieren la utilización de funciones trascendentales. Las unidades lineales rectificadas (ReLU) se definen de la siguiente forma:

$$y = f_{\text{relu}}(x) = max(0, x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases}$$

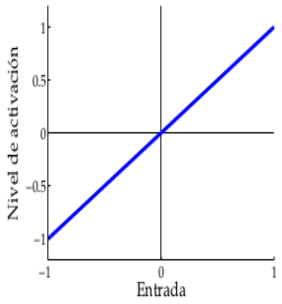
Existen otras variantes como Leaky Relu, ERelu, entre otras, pero no se usarán en este trabajo.

En general, la función de activación lineal rectificada tiene una ventaja con respecto a las sigmoidales cuando forman parte de una red que se entrena con algoritmos basados en el descenso por el gradiente. Las funciones sigmoidales se saturan a partir de cierto valor, lo que hace que la derivada sea prácticamente nula en casi todo su dominio. En el entrenamiento basado en back-propagation, se utiliza el valor de esa derivada como parte del cálculo del gradiente del error. Ese gradiente es el que determina cómo modificar los pesos de la red. Cuando la derivada de la función de activación es muy pequeña, las actualizaciones de los pesos serán muy pequeñas y el entrenamiento de la red avanzará muy lentamente.

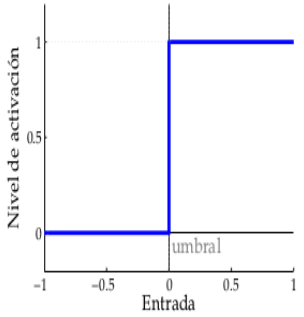
En la Figura1.2 pueden verse los gráficos de todas las funciones de activación recién mencionadas.

1.2.3. Redes neuronales artificiales

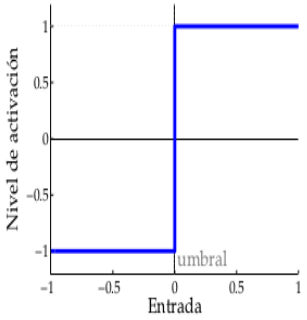
Un primer punto a tener en cuenta a la hora de abordar la construcción de cerebros artificiales como ensambles de neuronas artificiales, es asumir que el área de la neurociencia computacional está actualmente impregnada por el llamado *paradigma conexionista*, según el cual, los sistemas nerviosos naturales, incluidos los cerebros humanos, almacenan, procesan y recuperan información a través de la arquitectura de conexiones sinápticas, esas pequeñas regiones en las cuales el axón de una



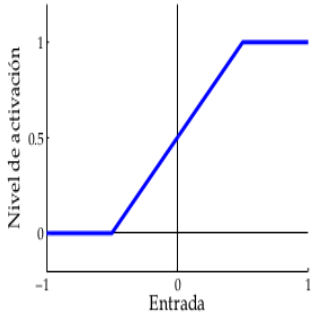
(a) Función de activación lineal



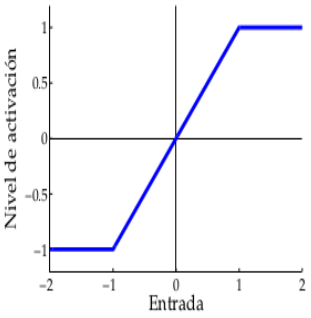
(b) Función de activación Heaviside



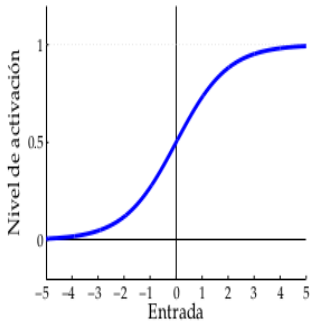
(c) Función de activación signo



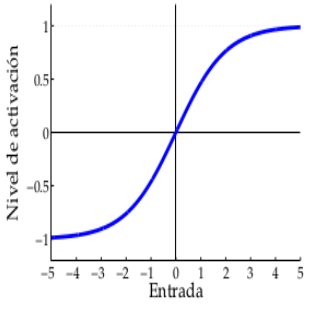
(d) Función de activación lineal con saturación



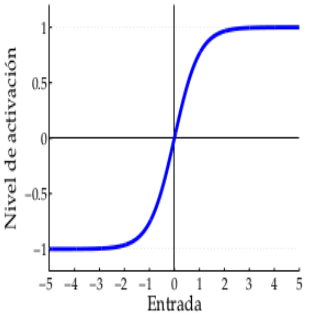
(e) Func. de activación lineal con saturación simétrica



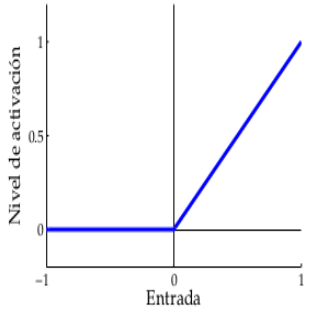
(f) Función de activación logística



(g) Función de activación sigmoidal bipolar



(h) Función de activación tangente hiperbólic



(i) Función de activación ReLU

Figura 1.2: Funciones de activación.

neurona pre-sináptica hace contacto con la dendrita de una neurona post-sináptica. Esta idea fue establecida por primera vez en el año 1949 por el psicólogo canadiense Donald Hebb en su libro *The organization of behavior* (La organización del comportamiento) Hebb (1949). Desde entonces ha jugado un papel clave en el desarrollo de la inteligencia artificial basada en neuronas artificiales. A lo largo de más de siete décadas, diversas disciplinas vinculadas a las neurociencias han confirmado repetidamente la validez del paradigma conexionista, el cual muchas veces recibe también el nombre de *paradigma hebbiano*, como reconocimiento a la agudeza de Donald Hebbs al tratar de entender el aprendizaje en términos celulares.

Construir una red neuronal artificial no es más que componer neuronas artificiales, cada una de las cuales viene determinada por su función de activación y su umbral. No obstante, a estas propiedades de las componentes es necesario agregarle también el modelado de la arquitectura de conexiones sinápticas, o sea todas y cada una de las sinapsis permitidas. Ya en el modelo de MaCulloch y Pitts las sinapsis fueron modeladas como números reales que toman valores positivos para representar sinapsis excitatorias, negativos para representar sinapsis inhibitorias y nulos para indicar la ausencia de conexión. En definitiva, aprender en términos conexionistas o hebbianos no es más que determinar, usualmente en forma algorítmica, cuál es el valor del umbral de cada neurona y el valor de cada conexión sináptica permitida entre las neuronas. Vale destacar que si no existieran sinapsis competitivas, o sea, excitatorias e inhibitorias, conviviendo en una red neuronal natural o artificial, estas no tendrían capacidad de computo.

En su topología más habitual las redes se organizan en grupos de neuronas llamadas capas, que se conectan entre ellas de tal forma que la salida de la capa i se utiliza como entrada en la capa $i + 1$. Estas redes reciben el nombre de redes neuronales feed-forward, al carecer de mecanismo alguno de realimentación (feedback). Las redes en las que si se incluye algún tipo de realimentación de las salidas a las entradas se denominan redes recurrentes. En este trabajo nos limitaremos a considerar redes feed-forward.

En estas redes existen dos tipos de capas: las capas visibles y las capas ocultas. Las primeras son aquellas que son observables desde el exterior de la red y consisten

de una *capa de entrada* y una *capa de salida*. La segunda categoría esta compuesta por todas las capas intermedias que no son observables desde afuera (por ello su nombre).

En las redes tipo feed-forward podemos encontrar diferentes arquitecturas dependiendo del número de capas ocultas que utilicen Hertz et al. (1991), a saber:

- Redes simples: Es el caso más simple y las redes solo tienen una capa de salida. En este caso la red se puede descomponer en muchas redes con una única neurona de salida. Si bien son fáciles de entender, son muy limitadas a la hora de aprender.
- Redes multicapa: Red neuronal simple, con una o pocas capas ocultas, además de las entrada y la salida. Este tipo de red ya fuerza al modelo a utilizar algoritmos de *backpropagation* para ajustar parámetros internos.
- Redes profundas [*deep networks*]: Redes que incluyen muchas capas ocultas.

En la fig. 1.3 presentamos una típica arquitectura feed-forward con dos capas ocultas. Observen que la entrada se puede pensar como un vector en $\mathbb{R}^6$, pues la capa de entrada tiene seis unidades. De la misma forma en este caso la salida se puede pensar como un vector en $\mathbb{R}$. De esta forma, podemos imaginar que la red artificial en cuestión debe *aprender* de la experiencia una función desconocida:

$$g : \mathbb{R}^6 \to \mathbb{R}$$

Queda claro a partir de la figura que la red es direccionada, no posee bucles y no hay conexiones entre neuronas de una misma capa.

1.2.4. Aprendizaje supervisado en redes feed-forward

Como hemos dicho anteriormente, en este trabajo nos limitamos a considerar aprendizaje supervisado en redes feed-forward. En otras palabras, vamos a suponer que nuestras redes son capaces de aprender de la experiencia, y que aprender significa precisamente determinar los valores de todas las conexiones sinápticas permitidas

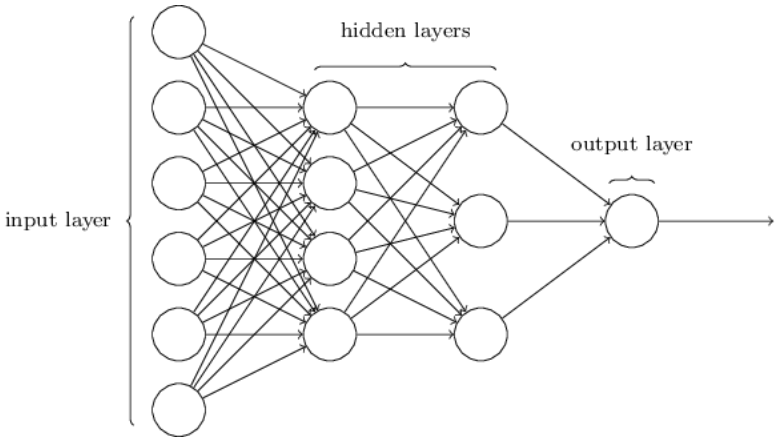


Figura 1.3: Esquema de una red feed-forward con dos capas ocultas.

entre las neuronas artificiales de nuestra red (recordemos que el umbral de cada neurona ha sido asimilado como una sinapsis más). La experiencia en este caso hace referencia a datos que podemos usar y que han sido previamente procesados por un experto (humano o no) capaz de asignar a p vectores de entrada $\vec{x^\alpha}$ ($\alpha = 1, 2, \ldots, p$) que pertenecen a $\mathbb{R}^N$ (donde N es el número de unidades de entrada) la salida supuestamente correcta $\vec{\hat{y}}^\alpha$ (la cual pertenece a $\mathbb{R}^M$, donde M es el número de unidades de salida). La experiencia de la cual aprenderá la red viene dada entonces por el llamado *conjunto de entrenamiento*, el cual es un conjunto de p elementos de la forma:

$$C = \{x^\alpha; \vec{\hat{y}}^\alpha\} \qquad \text{con} \quad \alpha = 1, 2, \ldots, p$$

1.2.5. Entrenamiento de la Red

En esta sección vamos a analizar el algoritmo más utilizado para entrenar redes neuronales, el cual fue introducido por D. Rumelhart, G. Hinton y R. Williams en el año 1986 Rumelhart et al. (1986) en un trabajo histórico del cual podemos afirmar, sin lugar a dudas, surge el actual desarrollo de la Inteligencia Artificial (IA). La idea consiste en ser capaz de asignar a cada sinapsis de la red un valor adecuado para que esta asigne correctamente las p salidas correctas $\vec{\hat{y}}^\alpha$ a cada una de las p entradas $\vec{x^\alpha}$ del conjunto de entrenamiento. En la próxima sección entonces presentaremos el algoritmo de back-propagation. Para cuantificar que tan bien se está logrando este objetivo, se define una función usualmente llamada de costo, de error o de

pérdida (loss function) $L(\{w_i\})$. Esta función mide la "diferencia" entre los valores de las salidas deseadas (etiquetas) y las salidas predichas por la red. La función de pérdida se calcula siempre a partir de la función de activación de las neuronas de la capa de salida, por lo tanto, el objetivo del algoritmo de entrenamiento será minimizar $L(\{w_i\})$ en función de los pesos sinápticos o parámetros del modelo. En otras palabras, se desea encontrar un conjunto de pesos que hagan que el error sea lo más pequeño posible. De momento, vamos a dejar de lado la forma especifica que puede tomar la función de pérdida y solo diremos que es una función de muchas variables que deseamos minimizar. Entonces, desarrollaremos primero la técnica que se utiliza para lograr ese objetivo y luego volveremos a la función de pérdida para especificar y desarrollar algunas en particular.

Back-propagation

El algoritmo de back-propagation permite, a partir de calcular el error que la red comete en el proceso de aprendizaje, determinar valores adecuado (no únicos) de cada una de las sinapsis permitidas por la arquitectura feed-forward. Para ello se eligen valores iniciales aleatorios de las sinapsis, usualmente con una distribución de probabilidad de media cero y desviación estándar igual a uno y se calcula el gradiente de la función de error con respecto a los diferentes parámetros de la red. El gradiente obtenido indica en qué sentido han de modificarse los parámetros. La actualización se realiza comenzando por la última capa de parámetros y se avanza hacia las primera pasando por todas las intermedias, en ese orden. Explicado esto, ahora entendemos de donde surge la necesidad de que la función de activación de las neuronas sea derivable. Una vez calculado esto, la técnica de optimización es la que realiza el ajuste de dichos parámetros con el objetivo de minimizar el error. Esta técnica basada en back-propagation es un ejemplo típico aprendizaje supervisado.

Para encontrar el camino hacia el mínimo de la función de costo empecemos pensando cómo se modifica esta cuando realizamos un pequeño cambio en los parámetros. Matemáticamente sabemos que

$$\Delta L \approx \nabla L \cdot \Delta w \tag{1.2.2}$$

donde $\Delta w = (\Delta w_1, ..., \Delta w_M)^T$, T es la operación de transposición, M es el número de parámetros del modelo y el gradiente ∇L se calcula como

$$\nabla L \equiv \left(\frac{\partial L}{\partial w_0}, ..., \frac{\partial L}{\partial w_M} \right)^T .$$

Lo interesante de la Ecuación 1.2.2 es que vemos claramente como una "buena" elección de Δw puede garantizar que recorremos adecuadamente el camino hacia el mínimo, si logramos que ΔL sea negativo. En particular, si suponemos que

$$\Delta w = -\eta \nabla L,$$

donde η es un parámetro positivo pero suficientemente pequeño llamado *tasa de aprendizaje* (learning rate) y remplazamos en 1.2.2, obtenemos:

$$\Delta L \approx -\eta \nabla L \cdot \nabla L = -\eta \|\nabla L\|^2$$

donde $\|\nabla L\|^2 \geq 0$, por lo que $\Delta L \leq 0$, es decir que L siempre va a disminuir. Esto provee una forma de seguir el gradiente al mínimo aplicando repetidamente la regla de actualización:

$$w \to w' = w + \Delta w$$

Para que el algoritmo de aprendizaje funcione correctamente es necesario hacer una buena elección de η. Debe ser lo suficientemente pequeño para garantizar que L decrece pero no tan pequeño que haga que los cambios en en Δw sean diminutos y la convergencia sea demasiado lenta. Es este motivo el que impulsó a automatizar la selección de la tasa de aprendizaje adecuada y, más aún, modificarlo en forma adaptativa a medida que el entrenamiento avanza. En su versión original el algoritmo de back-propagation usa siempre el mismo η, pero existen otros métodos adaptativos como Adagrad, Adadelta, RMSprop, Adam y NAdam que permiten alcanzar soluciones mucho mejores y en tiempo mucho más cortos. En este trabajo utilizaremos el método ADAM (por Adaptive Moment Estimation en inglés). A continuación mostramos la fórmula de actualización de dicho optimizador.

$$
\begin{aligned}
m &= \beta_1 \cdot m + (1 - \beta_1) \cdot \Delta w \\
v &= \beta_2 \cdot v + (1 - \beta_2) \cdot \Delta w^2 \\
w \rightarrow w' &= w - \frac{\alpha \cdot m}{\sqrt{v + \varepsilon}}
\end{aligned}
\tag{1.2.3}
$$

donde m y v representan los dos momentos, siendo m el que modela la media de los gradientes a lo largo del tiempo, mientras que v hace lo mismo con la varianza. β_1 es igual, en la mayoría de los casos, a 0,9, mientras que β_2 casi siempre se fija en 0,99. Vale aclarar que esta fórmula se aplica para cada sinapsis por separado, o sea, trata a cada dimensión del dominio de la función de pérdida de forma específica.

Habitualmente, recorreremos varias veces el conjunto de datos de entrenamiento para ajustar los parámetros de la red neuronal. Cada uno de esos recorridos recibe el nombre de *época*.

Existen tres opciones distintas en lo referido a la frecuencia con la que se ajustan los pesos durante el entrenamiento de una red neuronal: entrenamiento por lote (batch), entrenamiento en línea (on line) o entrenamiento por mini-lotes (mini-batch). Esta última es la forma que usaremos, en la cual el conjunto de entrenamiento es divido en Q lotes de tamaño l de forma tal que $p = Ql$ y en cada época se vuelve a sortear la partición a los efectos de introducir aleatoriedad en el descenso por el gradiente y evitar quedar atrapados en mínimos locales.

Usualmente en el entrenamiento de una red, junto con todas las técnicas recién mencionadas, también se usa la técnica de aleatoria de “dropout”. En pocas palabras, esta consiste en suprimir aleatoriamente en cada época un porcentaje determinado de neuronas y por ende todos los enlaces en las que ellas participan. Esto permite evitar el sobre-ajuste y hace más eficiente el descenso por el gradiente, agregando otro elemento aleatorio.

Función de pérdida

Existen varias funciones de pérdida que se suelen usar frecuentemente en la literatura para problemas de regresión y la mejor elección depende del problema

que se desea resolver. A continuación las dos que usaremos en este trabajo:

- *Valor medio absoluto* (MAE por su sigla en inglés) o L1: Es la suma sobre las neuronas de salida y sobre los elementos del mini-batch del valor absoluto de la diferencia entre el valor de activación de una neurona y su valor real o esperado. Algebraicamente se define como

$$E = \frac{1}{n} \sum_{\alpha} \left| y^{\alpha} - \vec{\hat{y}}^{\alpha} \right|,$$

  donde la suma corre sobre las neuronas de salida y sobre los elementos del conjunto de entrenamiento.

- *Error cuadrático medio* (MSE) o L2. Su valor se define como la suma sobre las neuronas de salida y sobre los elementos del mini-batch de los cuadrado de la diferencia entre el valor de activación de una neurona y su valor real o esperado. Algebraicamente se define como

$$E = \frac{1}{n} \sum_{\alpha} \left(y^{\alpha} - \vec{\hat{y}}^{\alpha} \right)^2$$

A la hora de hablar de problemas de clasificación, aunque el MSE fue más popular en los años 80 y 90, fue gradualmente reemplazado por funciones de costo basadas en la entropía cruzada, definida como:

$$L(\{w_i\}) = -\sum_{\alpha} [\hat{y^{\alpha}} log(y^{\alpha}) + (1 - \hat{y^{\alpha}}) log(1 - y^{\alpha})]$$

donde α corre sobre las neuronas de salida y el conjunto de elementos del mini-batch, y_{α} indica las salidas de la red y $\hat{y}_{\alpha}$ son las salidas deseadas. Esta función tiene su origen en teoría de la información y es una forma de medir la "distancia" entre dos distribuciones de probabilidad.

Cuando tenemos un problema de clasificación con más de dos clases, la capa de salida de nuestra red incluirá necesariamente varias neuronas. Cuando tenemos J clases diferentes utilizamos J neuronas de salida. Si dichas clases son disjuntas podemos interpretar los niveles de activación de cada neurona como la estimación

que hizo la red de la probabilidad de que el ejemplo presentado pertenezca a cada clase. Para hacer que la red neuronal represente dicha distribución de probabilidades se utiliza la función *softmax*:

$$y_j = \frac{e^{z_j}}{\sum_{i=1}^{J} e^{z_i}}$$

donde y_j es el nivel de activación de la j-ésima neurona de salida y z_j es su entrada neta ($z_j = \sum_i w_{ij} x_i$). Ésta función nos permite interpretar la salida i-ésima de la red como estimación de probabilidad de que la clase i sea la clase correcta para el ejemplo correspondiente de entrada. En otras palabras, es una forma de reescalar los vectores correspondientes a las entradas netas de las neuronas de la capa de modo que el resultado pueda interpretarse como una distribución de probabilidad.

Conjunto de entrenamiento

El conjunto de entrenamiento es una componente crítica para el rendimiento del modelo. Es fundamental que sea completo y representativo de los datos reales que encontrará la red una vez entrenada y puesta en marcha en el entorno real. Además, suele ser recomendable que el conjunto cumpla las siguientes características: cada variable de entrada debería tener una media cercana a cero, una escala ajustada para que las varianzas seas similares y, dentro de lo posible, ser descorrelacionadas.

Una de las dificultades más frecuentes de los algoritmos de aprendizaje profundo es que necesitan ser entrenados en conjuntos de datos muy grandes para funcionar de manera efectiva. Pero, una forma simple de conseguir un rendimiento mejor es mediante el proceso de aumento de datos (data augmentation), el cual añade al conjunto de entrenamiento datos creados artificialmente. En particular es una herramienta muy útil en aplicaciones de redes convolucionales con imágenes ya que se incluyen una enorme variedad de factores a tener en cuenta y de muy fácil simulación. Se suele ampliar el conjunto de datos usando ciertas transformaciones de los mismo. Los más comunes suelen ser rotaciones, cambios de escala y traslaciones (Goodfellow et al. (2016)). Esto sirve para aumentar artificialmente la experiencia de la red durante el entrenamiento, exponiéndola a variaciones en los datos que podría llegar a encontrar en la práctica real. Por ello es importante identificar qué

transformaciones pueden ser de utilidad en función de la tarea que queremos que aprenda la red y la variabilidad en los datos con los que va a trabajar.

1.2.6. La evaluación del desempeño de las redes para clasificación

En esta sección analizaremos cómo evaluar el desempeño de las redes. Como dijimos, la implementación del algoritmo de back-propagation requiere que definamos la función de pérdida, como métrica del error cometido, tanto en el proceso de entrenamiento como en el posterior funcionamiento. Para ello del conjunto de entrenamiento se reserva una fracción (usualmente pequeña, típicamente del 10 %) que se usa para verificar la capacidad de generalizar la regla aprendida a los ejemplos nunca vistos. Este subconjunto del conjunto de entrenamiento se denomina *conjunto de testeo*. Además durante el entrenamiento, es importante monitorear el desempeño de la red en un subconjunto de datos que no se usa para el entrenamiento. Esto se debe a que los algoritmos de aprendizaje automático tienden a funcionar mejor con los datos con los que han sido entrenados. Al realizar pruebas periódicas con datos no vistos, podemos asegurarnos que la red generalice bien la tarea fuera del conjunto de entrenamiento. El subconjunto de datos utilizado para esto se conoce normalmente como el *conjunto de validación*. Notemos que ahora el conjunto de entrenamiento ha sido reducido. Es importante que esta partición del conjunto de datos sea aleatoria.

Para el caso de regresión la función de error es una medida adecuada de caracterización del desempeño. En cambio, para el caso de clasificación el problema es más delicado. Para comenzar, toda clasificación presupone primero una regresión y utilizando las función *softmax* es posible clasificar sin tener que recurrir a funciones de activación discretas en la salida, las cuales imposibilitarían el uso del descenso por el gradiente.

Los indicadores de desempeño son muy útiles a la hora de evaluar y comparar diferentes modelos y técnicas de aprendizaje automático. Hay muchas métricas que pueden usarse para probar la capacidad de cualquier clasificador multi-clase y resultan útiles para: i) comparar el rendimiento de dos modelos diferentes y ii) anali-

zar el comportamiento del mismo modelo ajustando diferentes parámetros. Muchas métricas se basan en la *Matriz de Confusión*, ya que encierra toda la información relevante sobre el algoritmo y rendimiento de la regla de clasificación.

Matriz de Confusión

La matriz de confusión para el caso de clasificación es una matriz $J \times J$ (donde J es la cantidad de categorías de clasificación) en la cual las columnas representan las predicciones del algoritmo y las filas las etiquetas reales de cada entrada. Por lo tanto, un elemento arbitrario $M_{i,j}$ (fila i y la columna j) de la matriz contiene el conteo de cuantas veces el sistema predijo clase j cuando la etiqueta real pertenecía a la clase i. De esta forma la diagonal principal de la matriz representa los elementos correctamente clasificados Grandini et al. (2020) y los elementos fuera de la diagonal la cantidad de errores cometidos. También es usual usar la fracción de veces que se tuvo cada tipo de resultado en lugar de usar el conteo. A estas matrices se las llama normalizadas.

Métricas

Con los datos obtenidos a partir de la dicha matriz se pueden calcular gran variedad de métricas como las mencionadas a continuación. La más utilizada en problemas de clasificación es la exactitud ("accuracy" en inglés). Esta se calcula como la proporción entre las predicciones correctas que ha hecho el modelo y el total de predicciones.

$$Exactitud = \frac{\sum_i M_{i,i}}{TotalCasos}$$

Sin embargo, aunque en ocasiones resulta práctico por su facilidad de cálculo, otras veces es necesario profundizar un poco más y tener en cuenta los tipos de predicciones correctas e incorrectas que realiza el clasificador.

Como en esta tesina se trabaja con clasificaciones en muchas clases vamos a definir las siguientes métricas siempre para una clase k dada, sabiendo que pueden hacerse para las J clases involucradas. Cuando se hace este cálculo por clase las

métricas se nombran "MACRO".

La precisión se refiere a lo cerca que está el resultado de una medición del valor verdadero. Se representa por la proporción entre los positivos reales predichos por el algoritmo y todos los casos positivos de dicha clase. Se calcula según la ecuación:

$$Precision_k = \frac{M_{k,k}}{\sum_i M_{i,k}}$$

La sensibilidad ("Recall" en inglés) y la especificidad son dos valores que nos indican la capacidad de nuestro estimador para discriminar los casos positivos de los negativos. Mientras que la especificidad, es la fracción de verdaderos negativos, la sensibilidad es la fracción de elementos bien clasificados de cierta clase dividida por el número total de unidades de esa clase. Mide la precisión predictiva del modelo para cierta clase.

$$Sensibilidad_k = \frac{M_{k,k}}{\sum_j M_{k,j}}$$

Combinando dos métricas se calcula la F1-Score (macro). Esta puede ser interpretada como un promedio pesado entre la precisión y la sensibilidad. Lo primero que se debe hacer es calcular un promedio macro de cada una de las anteriores de la siguiente forma

$$PromedioPrecisionMacro = \frac{\sum_k Precision_k}{J} \text{ (MAP)},$$

$$PromedioSensibilidadMacro = \frac{\sum_k Recall_k}{J} \text{ (MAR)},$$

y con estos dos valores la MACRO F1-SCORE se calcula como en la siguiente ecuación:

$$F1_{macro} = \frac{2}{MAP^{-1} + MAR^{-1}}$$

La métrica obtenida evalúa el algoritmo desde el punto de vista de la clase: los valores altos de $F1_{macro}$ indican que el algoritmo tiene un buen desempeño en todas las clases, mientras que los valores bajos se refieren a la existencia de clases mal predichas.

1.3. Redes Neuronales Convolucionales

En este trabajo vamos a utilizar una familia especial de redes feed-forward llamadas Redes Neuronales Convolucionales.

Las redes neuronales convolucionales (CNN en inglés por Convolutional Neuronal Network), introducidas por LeCun et al. (1995), son un tipo particular de redes neuronales artificiales que se caracterizan por hacer uso de la *operación de convolución*. Estas redes se utilizan habitualmente para resolver problemas de procesamiento de señales, particularmente imágenes. La principal diferencia con una red multi-capa convencional radica en que tanto sus datos de entrada como de salida pueden ser estructurados, es decir pueden ser vectores $1D$ (señales de audio), matrices $2D$ (píxeles de una imagen) o tensores ND (imágenes a color, canales RGB), en el que la relación espacial en las entradas es muy relevante. La esencia del método consiste en utilizar la topología (en nuestro caso bidimensional) de la entrada en el aprendizaje. Esta clase de redes utiliza tres ideas básicas: campos receptivos locales, pesos compartidos y pooling. A continuación explicamos cada idea por separado.

En la Figura 1.3 se mostró una red con capas totalmente conectadas (fully-connected), es decir donde todas las neuronas de una capa se conectan con todas las neuronas de la capa siguiente y, mientras que las entradas estaba representada como un vector $N-dimensional$, en las CNN se deja la entrada con la topología del dato mismo. Por ejemplo, si trabajamos con imágenes en tonos de grises, como en esta tesina, la entrada será una matriz bidimensional de HxL neuronas, las cuales corresponden a las dimensiones (alto y ancho) de píxeles de la imagen que vamos a procesar. Entonces la idea de campo local receptivo refiere a que no vamos a conectar cada píxel de entrada a todas las neuronas de la capa oculta siguiente. En cambio dejaremos solo conexiones en una región pequeña y localizada de la imagen de entrada. En otras palabras, cada neurona de la primera capa oculta se conectara con una pequeña región de las neuronas de entrada, por ejemplo de kxk neuronas. Así, para una neurona particular de la capa oculta, tendremos conexiones que se verán como en la Figura1.4.

Dicha región de la imagen de entrada es llamada *campo local receptivo* de la

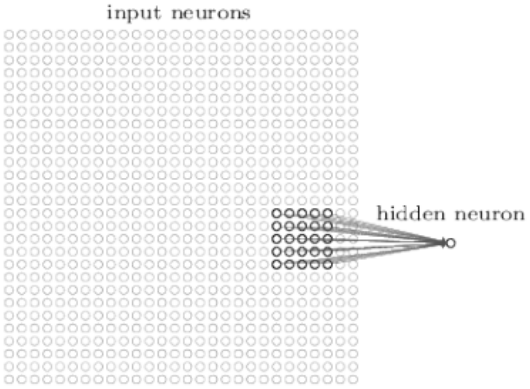


Figura 1.4: Esquema de conexiones de capas convolucionales con campos locales.

neurona de la capa oculta. Para completar la construcción de la primera capa oculta dicha región o ventana se desliza a través de la imagen completa y de cada región hay una neurona oculta en la primera capa como se muestra en la Figura 1.5. Notemos que el tamaño de la capa oculta tiene menor dimensión que la de entrada dado que dependiendo del tamaño de la imagen y de la región elegida solo podemos recorrer cierta cantidad de pasos sin salirnos de los límites de la imagen. Por ejemplo si tenemos una imagen de $28x28$ píxeles y una ventana de $5x5$ la capa de salida contendrá solo $24x24$.

Como ya se mencionó, cada neurona de la capa oculta tiene kxk pesos conectados al campo receptivo local. Lo interesante de esta clase de redes es que se usan los mismos pesos para todas ellas. Esto significa, como veremos más adelante, que todas las neuronas en la capa detectan las mismas características pero en diferentes ubicaciones de la imagen de entrada y por esto tienen la capacidad de ser invariantes ante traslaciones. Es por esta razón que a la salida de una capa convolucional creada a partir del barrido con una ventana con los mismos pesos se suele llamar *mapa de características* (feature maps). Esta característica es la que llamamos *pesos compartidos* (shared weights) y al conjunto de pesos compartidos se los define como un *filtro* (kernel). Una gran ventaja de compartir los pesos es que reduce significativamente el número de parámetros involucrados en las redes.

Además de estas características, las redes convolucionales también hacen uso de las llamadas capas de pooling de las cuales hablaremos más adelante (1.3.3).

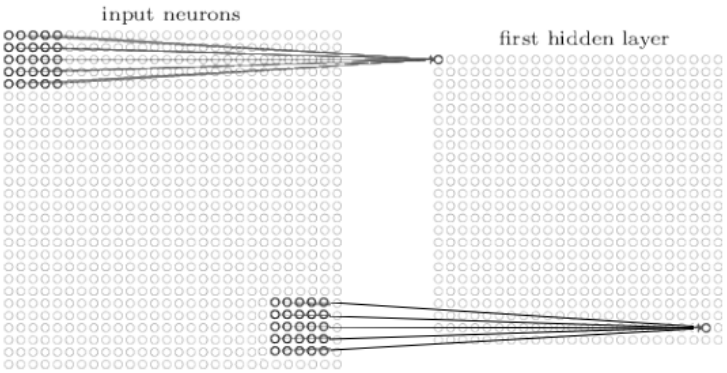


Figura 1.5: Construcción de la siguiente capa.

El problema o tarea de una red puede consistir en: i) clasificar (utilizando una capa softmax), ii) en hacer regresión o iii) en aprender una red auto-encoder, reflejando en la salida la topología de la entrada. En particular, como en esta tesina trabajaremos con imágenes en tonos de grises, nos interesa el caso de auto-encoders que tienen imágenes de salida del mismo tamaño que las de entrada. Antes de continuar con esta arquitectura en particular explicaremos como funcionan las capas convolucionales y sus filtros.

La entrada de una capa convolucional se procesa realizando una *convolución* con un kernel cuyos valores serán los parámetros que aprenderá la red durante el entrenamiento. Definamos entonces la operación de convolución.

1.3.1. Operación de convolución

La operación de convolución es una operación matemática que se define como la integral del producto de dos funciones después de que una de ellas se refleje y desplace. Usualmente las funciones involucradas en la operación son la señal que deseamos procesar $\zeta[n]$ y un filtro (kernel), al que denotaremos como $K[k]$. Si tratamos con señales digitales discretas, como es nuestro caso, la integral se sustituye por una sumatoria. En particular para señales bidimensionales en las que aplicamos un filtro de convolución $K[k_1, k_2]$, la convolución se calcula utilizando Ecuación 1.3.1:

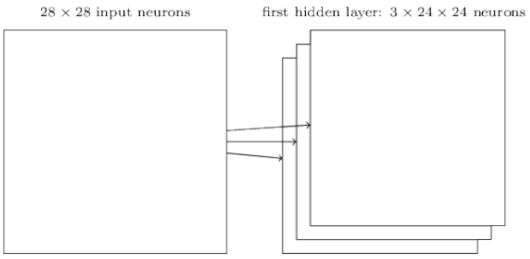


Figura 1.6: Salida de una capa convolucional con 3 filtros.

$$(\zeta \star K)[n_1, n_2] = \sum_{k_1=0}^{K_1-1} \sum_{k_2=0}^{K_2-1} K[k_1, k_2]\, \zeta[n_1 - k_1, n_2 - k_2] \tag{1.3.1}$$

1.3.2. Capas convolucionales

Como su nombre lo indica estas capas son las que se encargan de realizar la operación de convolución a sus entradas utilizando filtros. Como ya mencionamos anteriormente, a diferencia de lo que sucede en las capas totalmente conectadas, las neuronas de una capa convolucional solo se conectan a una región local de la entrada y comparten sus pesos, por lo que cada neurona que se emplea detecta una característica local de la señal de entrada del campo receptivo. Dado que se requiere un conjunto completo de neuronas para detectar una única característica en distintas posiciones de la entrada, en la práctica, en una capa convolucional se incluyen distintos tipos de detectores de características, todos aprendidos, correspondientes cada una a un filtro concreto. Como la salida de cada tipo de detector es, a su vez, una señal del mismo tipo que la señal de entrada, se suele hablar de mapas de características (*feature maps*). Cada tipo de detector generará un mapa diferente, por lo que la salida de una capa convolucional capaz de detectar S características diferentes estará formada por S mapas de características. En la Figura 1.6 puede verse un ejemplo de una capa con 3 mapas de características, donde cada mapa esta definido por un filtro de $5x5$ pesos compartidos. En la práctica se suelen usar muchos más detectores de características por capa.

En la Figura 1.7 se puede apreciar gráficamente un ejemplo de la aplicación de

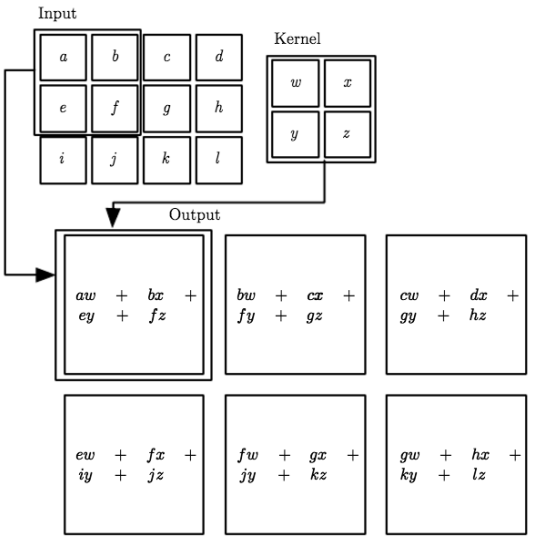


Figura 1.7: Esquema de aplicación del filtro a una entrada.

un filtro a una imagen bidimensional. En este caso el filtro está definido como una matríz 2×2. Cada punto de la salida es el resultado de la aplicación del filtro a una región de la imagen de entrada. Como se restringe la operación solamente a la aplicación del filtro dentro de los límites de la imagen de entrada, el mapa de características resultante tiene dimensiones estrictamente menores a esta.

Hiper-parámetros de las capas convolucionales

Al diseñar una capa convolucional existen 4 valores a determinar. Dichos valores se denominan *hiper-parámetros*, para diferenciarlos de los parámetros que aprende la red, ya que son valores que se fijan desde fuera de la red antes de comenzar el entrenamiento. Dos de ellos ya han sido mencionados; el **número de filtros** (profundidad) y el **tamaño** K (conectividad local) de los mismos. Los dos restantes son el uso de **rellenado** (padding) y su **paso** (stride).

- *Paso*: Establece un tamaño (*paso*) que indica cada cuantas muestras de la entrada se pide una salida. En el caso de imágenes, si se utiliza un $paso = (1, 1)$ (uno para cada dimensión) éste realizaría una convolución tradicional píxel a píxel, en cambio si usamos $paso = (2, 2)$ haría una convolución sub-muestreada que generaría una salida 4 veces más pequeña que la entrada original. Un ejemplo de lo recien mencionado es ejemplificado en la Figura 1.8.

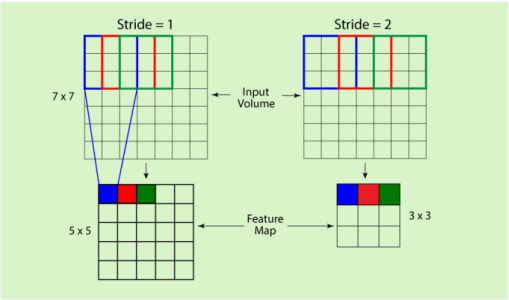


Figura 1.8: Ejemplo de diferentes valores de paso.

- *Rellenado*: Fija la cantidad de relleno, Z, de la entrada alrededor para obtener una salida del tamaño deseado. Es decir un rellenado de $Z = (l, l)$ agrega l filas y l columnas en los bordes horizontales y verticales superior e inferior de la imagen. El relleno más utilizado es el *zero padding* que, como su nombre indica, agrega ceros, pero existen otros tipos de relleno como continuar el borde, espejar la imagen, etc.

En la Figura 1.9 vemos el efecto en la operación de convolución al usar rellenado $Z = (l, l)$ y un paso $paso = (1, 1)$.

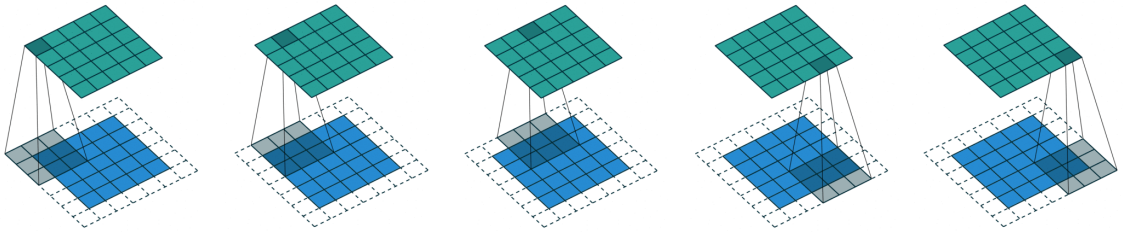


Figura 1.9: Ejemplo de construcción de la capa oculta usando rellenado y paso

1.3.3. Capas de Pooling

Típicamente una capa convolucional esta compuesta de tres pasos. El primero, ya mencionado, es donde se llevan a cabo las convoluciones para producir un conjunto de activaciones. El segundo paso, llamado el paso de detección, consiste en pasar dichas activaciones por una función de activación no lineal (por ejemplo ReLU). El último consiste en usar una función no paramétrica, llamada función de pooling.

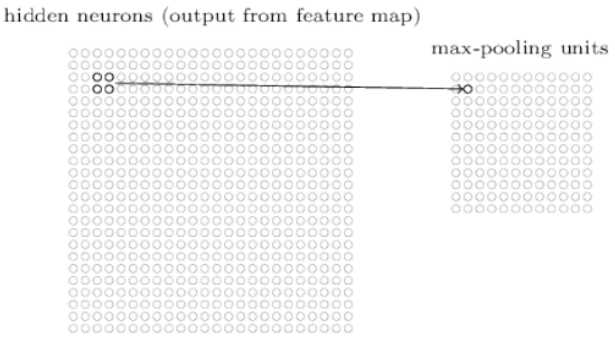


Figura 1.10: Esquema de como funciona una capa de pooling.

La función de la capa de pooling es reducir la dimensión espacial de los datos o su profundidad. En este trabajo solo se aplicará a la imagen pero no a la profundidad del tensor resultante (esto se llama *spatial pooling*). Para ello se utiliza, al igual que la capa de convolución, un filtro con cierta dimensión y paso. La forma más usual de aplicar pooling es la *Max pooling*, que tiene como salida el máximo de los valores de la ventana que usa. Otro muy común es el *Average pooling* que toma el promedio. En la Figura 1.10 vemos una capa de pooling con filtro 2×2 por lo que el tamaño del la entrada de dicha capa se reduce a la mitad, en ancho y alto, a la salida, ya que de cada 4 neuronas solo elige la de mayor valor de salida.

1.3.4. Auto-encoders

Como ya se ha mencionado, la salida de una CNN puede representar cierta estructura deseada. Para tratar estos casos vamos a introducir una arquitectura de red llamada *auto-encoders*.

Los *auto-encoders* son redes neuronales con un algoritmo de compresión de datos en el que las funciones de compresión (encoder) y descompresión (decoder) son especificas de los datos. Están entrenados para replicar la entrada de la red a la salida con el menor error posible. El encoder es una red neuronal que transforma sus entradas en una representación interna (intermedia), normalmente de dimensión inferior a la entrada, con el fin de obligarle a aprender una compresión eficiente que extraiga las propiedades principales de los datos de entrada. Posteriormente, esta representación intermedia (salida del encoder) se utiliza como entrada del decoder

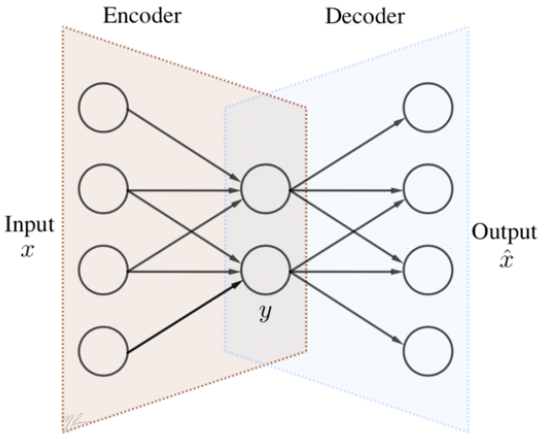


Figura 1.11: Esquema de la arquitectura de una red Auto-encoder.

para recuperar, en la medida de lo posible, la entrada original.

- **Encoder**: la parte de la red que comprime la entrada en un espacio de variables latentes y que puede representarse mediante la función de codificación $h = f(x)$.
- **Decoder**: la parte que trata de reconstruir la entrada en base a la información recolectada previamente en el espacio latente. Se representa mediante la función de decodificación $\hat{x} = g(h)$.

$$\text{input}_{\ 1} \in \mathbb{R}^n \rightarrow \text{output}_{\ 1} = \text{encoder}\ (\text{input}_{\ 1}) \in \mathbb{R}^m, m < n$$
$$\text{input}_{\ 2} = \text{output}_{\ 1} \rightarrow \text{output}_{\ 2} = \text{decoder}\,(\text{input}_{\ 2}) \in \mathbb{R}^n.$$

Debido a esto, se podría usar la salida del encoder como nueva representación del dato para cualquier tarea, no solo para reproducirlo, porque en esa representación se encuentra codificada toda la información relevante del dato de entrada para poder llevar a cabo su reconstrucción. De hecho, es lo que hacen todas las redes neuronales en cualquiera de sus capas intermedias, pero los auto-encoders centran su tarea en la capacidad reproductiva de la entrada, o de ligeras variantes de ella.

Espacio Latente

Cuando este proceso de auto-configuración ha terminado, lo que se obtiene es una herramienta capaz de producir imágenes muy similares a las que le hemos proporcio-

nado, pero que no se limitan a ellas, sino que podría generar variaciones infinitas. Las imágenes de entrenamiento forman parte, hipotéticamente, de las imágenes que puede generar la red entrenada, pero entre cada una de estas imágenes hay un número infinito de variaciones a la vez similares y diferentes.

El espacio de representaciones internas generado es lo que se llama *espacio latente*. Se lo puede pensar como un espacio de dos dimensiones donde las imágenes más similares estuvieran próximas entre ellas y donde, en consecuencia, se podría transitar entre dos imágenes cualesquiera transformándolas lentamente una en la otra a partir de pequeñas diferencias. Una red entrenada contiene, en cierta manera, un espacio como este, pero no compuesto por dos dimensiones sino por muchas más.

1.4. Descripción del Problema

En la presente tesina se trabajará con diferentes arquitecturas de redes neuronales convolucionales con el objetivo de aprender a reconocer ángulos de rotación de imágenes de entrada.

En un principio el problema será reducido a una clasificación, esperando que la red aprenda a discriminar rangos de ángulos de rotaciones de imágenes previamente rotadas y etiquetadas por nosotros, presentadas como diferentes clases. El siguiente paso será entrenar una red regresional con el objetivo de que aprenda a distinguir el ángulo de rotación que presenta la imagen de entrada. Nuevamente dicha etiqueta será puesta por nosotros. Finalmente, se entrenará un auto-encoder con la tarea de que, dada una imagen de entrada previamente rotada, la red sea capaz de reproducirla pero orientándola correctamente. Lo interesante de esta parte del trabajo es que usaremos un auto-encoder pero para entrenarlo utilizaremos aprendizaje supervisado, al brindarle, como salida esperada luego de la decodificación, la imagen original sin rotar.

Este proyecto se enmarca en estudios previos realizado en el grupo y que culminaron en las tesis de doctorado en Física de las Dras. Carolina Tauro y Carolina Daza y las correspondientes publicaciones. En esos trabajos se modeló la región V1 de la corteza visual en mamíferos, la cual tiene la función de reconocer orientaciones de

rotación en las imágenes que llegan a través de la retina. Este trabajo es parte de un proyecto más grande que busca reunir técnicas de modelado neuronal y de aprendizaje automático a los fines de estudiar la formación de patrones espacio-temporales en la corteza cerebral al realizar la tarea de identificar ángulos en la imagen rotada.

A continuación se presentan las tecnologías, herramientas y conjunto de datos a utilizar en el presente trabajo.

1.5. Lenguaje de Programación

El lenguaje de programación empleado ha sido Python, por ser el más utilizado en el campo de la inteligencia artificial y más concretamente en aprendizaje profundo (Deep Learning). Una de las principales ventajas que ofrece este lenguaje de programación es la amplia variedad de librerías desarrolladas para problemas de ciencia de datos e inteligencia artificial, las cuales son universalmente utilizadas hoy, tanto en el ámbito académico como en el sector privado. Algunas de las librerías utilizadas en en este proyecto son:

- PyTorch, detallada en la sección 1.5.1.
- Numpy, librería de código abierto que facilita la computación numérica con Python.
- Python Imaging Library (PIL), librería de código abierto para Python que proporciona potentes capacidades de procesamiento de imágenes.
- Optuna, Un marco de optimización de hiperparámetros de código abierto para automatizar la búsqueda de hiperparámetros

1.5.1. PyTorch

PyTorch es una librería de aprendizaje automático de código abierto para realizar cálculos numéricos haciendo uso de la programación de tensores. Además permite su ejecución en GPU (Graphic Processor Unit) para acelerar los cálculos. PyTorch fue principalmente desarrollado por Facebook AI e introducido en 2016. Se usa para la investigación y desarrollo en el campo del aprendizaje automático, centrado principalmente en el desarrollo de redes neuronales. Algunas alternativas a PyTorch también muy populares en el mundo académico son TensorFlow y Keras (desarrolladas y mantenidas por Google), entre muchas otras.

1.6. Conjunto de Datos

Lo primero que tenemos que pensar a la hora de entrenar redes neuronales es el conjunto de datos que vamos a utilizar y las salidas esperadas que queremos obtener, en otras palabras, el *conjunto de entrenamiento*, ya que cómo hemos dicho, este trabajo analizará solo técnicas de aprendizaje supervisado. Dicho conjunto será dividido en conjunto de entrenamiento, de validación y de evaluación. En nuestro caso vamos a trabajar con redes convolucionales por lo que el formato de los datos de entrada es un conjunto de imágenes bidimensionales, pues son imágenes en tonos de grises. La salida, en cambio, dependerá de la tarea que pretendamos que la red aprenda. Cada imagen viene con el resultado correcto de su clasificación, y diremos, como es común en el área, que las imágenes tienen una *etiqueta* con la respuesta correcta o son *etiquetadas*. El formato de las etiquetas que asociamos a cada ejemplo serán explicadas en los siguientes capítulos, cuando hablemos de las diferentes redes utilizadas (clasificación, regresión y auto-encoder).

En este trabajo se utilizará el conjunto de datos MNIST, conjunto muy popular y muy utilizado para el entrenamiento de redes neuronales por lo que muchas de las librerías en Python (en particular Pytorch) ya tienen un conjunto de comandos (script) predefinido para cargarlo. El conjunto de datos MNIST es una buena opción para entrenar modelos si se desea crear rápidamente un prototipo de una nueva idea porque el tamaño de las imágenes es pequeño y su contenido es relativamente simple. Esto significa que no hace falta usar redes muy complejas con muchas capas que tomarían mucho tiempo para entrenar para clasificarlas.

1.6.1. Conjunto MNIST

El conjunto de datos MNIST contiene imágenes escaneadas de dígitos escritos a mano acompañados de su correspondiente clasificación (el dígito que se quiso escribir). Su nombre proviene del hecho de que es un subconjunto modificado de dos conjuntos de datos recopilados por el National Institute of Standards and Technology (NIST) de Estados Unidos de Norteamérica. El conjunto esta dividido en dos partes, el conjunto de entrenamiento y el de evaluación. El primero contienen 60000

imágenes y el segundo 10000. Las imágenes están en escala de grises y tienen una dimensión de 28×28 píxeles. Las imágenes de dígitos escritos a mano se obtuvieron de 250 personas. Cabe destacar que, para lograr un mejor desempeño de testeo, el conjuntos de 250 personas del que se tomaron muestras para el conjunto de entrenamiento, es distinto del que se utilizo para evaluación, pero ambos formados por empleados del US Census Bureau y estudiantes de secundarios voluntarios.

Dado que este conjunto de datos no posee subconjunto de validación lo que se hizo fue separar una parte del conjunto de entrenamiento y se destino para crear el subconjunto de validación, dejando un 80 % para entrenamiento y el 20 % restante para validar.

Como el objetivo de la tesina es trabajar con imágenes rotadas, para poder utilizar el conjunto de datos MNIST, que son ejemplos de imágenes de dígitos escritos a mano que están orientados de manera "correcta", es decir derechos, y que nuestra red pueda tener como dato de entrada imágenes con diferentes ángulos de rotación, se modificará el conjunto para crear ejemplos que nos sirvan. Para ello crearemos, de cada imagen del conjunto de datos, tanto en entrenamiento como en validación y evaluación, 20 imágenes nuevas a partir de rotaciones distintas y aleatorias.

Para crear las rotaciones se utiliza el módulo *torchvision.transforms*. En particular, se utiliza el módulo de las transformaciones funcionales. Estas brindan un control detallado de la transformación. Las transformaciones funcionales no contienen un generador de números aleatorios para sus parámetros, lo que significa que se debe especificar/generar todos los parámetros. En nuestro caso el parámetro que se le pasa es el ángulo en que se desea rotar la imagen. Para hacer eso se crea una variable entera "ang" aleatoriamente, $ang = random.randint(min, max)$, donde min y max son los límites del intervalo en el cual queremos que se generen los números y se la pasa a la función "rotador = torchvision.transforms.functional.rotate" junto con la imagen que se desea rotar. Cabe destacar que la transformación funcional brindará resultados reproducibles en todas las llamadas.

1.7. Modelos

Como ya hemos explicado, en esta tesina buscamos diseñar tres tipos distintos de redes convolucionales. Dichos modelos, con sus objetivos, arquitecturas y resultados obtenidos serán explicados en capítulos independientes.

Capítulo 2

Clasificación

Como dijimos anteriormente lo primero que se realiza es una clasificación. La salida deseada de la red será la etiqueta que nos dice a que clase pertenece el ejemplo procesado, o sea, el dígito de entrada. Recordemos que, el problema a clasificar no es el dígito de entrada, sino el ángulo de rotación de la imagen de entrada con respecto a la figura original en MNIST. Al tratar el problema de esta forma, la red debería ser capaz de producir, a la salida, un vector de dimensión igual a la cantidad de clases que deseamos clasificar, donde cada elemento de dicho vector contiene el valor de probabilidad entre 0 y 1 de que el ejemplo pertenezca a esa clase.

En este trabajo consta de varias clasificaciones distintas, teniendo en cuenta el rango de ángulos de rotación y la cantidad de clases diferentes con las que cuenta. La arquitectura de las redes, para todos los casos, consta de dos capas convolucionales con filtros de 3×3 con función de activación ReLU, seguidas de una capa de pooling 2×2 tipo MaxPool y finalmente dos capas lineales totalmente conectadas con 128 y M neuronas, donde M es la cantidad de clases del correspondiente problema de clasificación. Se utiliza la técnica de dropout al finalizar la capa de pooling y luego de la primera capa lineal, utilizando un valor de $p = 0{,}20$. En esta parte del trabajo la función de costo utilizada es la entropía cruzada (cross-entropy loss). A continuación explicamos las diferentes versiones, sus restricciones y como se trataron los datos de entrada para cada red así como los resultados obtenidos de cada una.

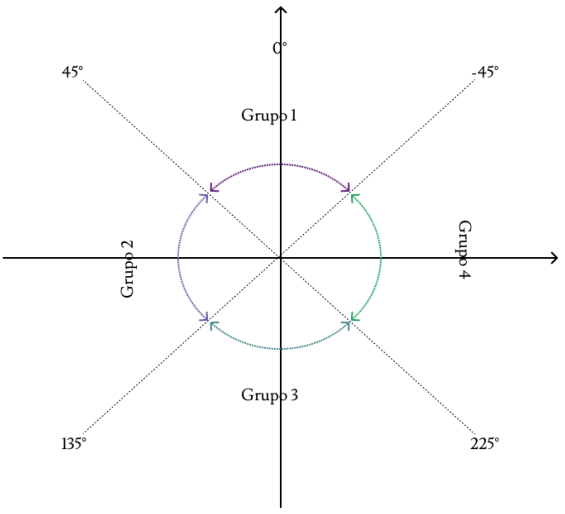


Figura 2.1: Gráfico de las diferentes etiquetas de clasificación para la red clasificadora 360°.

2.1. Clasificación en 360°

2.1.1. Clasificación en 4 clases

Lo primero que se hizo es una red para clasificar en 4 grupos distintos de ángulos de rotación que abarcaban los 360°, por lo que cada grupo contemplará 90°. Las etiquetas correspondientes a cada grupo irán desde el "grupo 1" cuyo rango es $[-45, 45]$ hasta el "grupo 4" con rango $[225, 315]$ (Figura 2.1).

Como hemos dicho, a cada imagen del conjunto MNIST se le realizan 20 rotaciones, cada una con un ángulo de rotación distinto y se le asigna la etiqueta de clase que corresponde, dependiendo del rango dentro del cual se encuentra dicho ángulo. En particular en esta parte se crean 5 imágenes para cada uno de los grupos antes mencionados, asegurándonos así, tener una distribución de datos de entrada equilibrada. La etiqueta de cada imagen nueva es, por lo tanto, el grupo al que pertenecía. Para comenzar decidimos que solo se utilizaría un conjunto reducido del conjunto MNIST completo dado que estamos trabajando con rotaciones que abarcan todos los ángulos de rotación posibles y ciertos números pueden llegar a confundir a la red por presentar simetrías rotacionales, como puede ser el cero, el ocho, el seis y el nueve. Como primer intento entonces se elige trabajar solo con imágenes que en el

Figura 2.2: Imagen original y un ejemplo de rotación de cada clase con su respectiva clasificación.

MNIST están catalogadas como 3, 4 y 7. En la Figura 2.2 se muestra una imagen de un dígito del subconjunto elegido con un ejemplo de rotación de cada clase y su respectiva etiqueta.

Antes de comenzar con el entrenamiento se deben elegir los mejores hiper-parámetros de entrenamiento, es decir, aquellos parámetros del modelo que no son aprendidos automáticamente, sino que son parte del modelo neuronal, tales como la tasa de aprendizaje, el momento, el número de filtros de cada capa convolucional, el peso de regularización y el tamaño de los mini-lotes, entre otros. Después de varias pruebas se decidió definir el tamaño de los mini-lotes de 5000 imágenes.

Por otro lado, los valores del número de filtros a utilizar en las capas convolucionales (nf), el algoritmo de actualización (optimizador), el peso de regularización (wr) y la tasa de aprendizaje (η) son elegidos a través de la utilización de la librería Optuna, mencionado en el capítulo anterior. Los valores que se inspeccionan son entre 16, 32 y 64 filtros, los métodos de descenso por el gradiente SGD o ADAM y para los valores de wr y η entre $1e-5$ y $1e-1$. A la hora de utilizar esta librería, siempre tenemos que definir tres componentes principales: el objetivo, el estudio y la optimización. El objetivo es la función que deseamos optimizar que depende de los parámetros. En la definición del estudio se definen los parámetros a evaluar dentro de la red y se da la "directiva" de optimización. En nuestro caso la función objetivo es la función de costo y la directiva es minimizar el valor de salida de dicha función. Una vez que tenemos el estudio y el objetivo definidos, se procede a realizar la optimización. En nuestro caso la optimización se llevó a cabo a lo largo de 200 ensayos y limitando el tiempo de ejecución a 6 horas.

En este estudio de optimización los hiper-parámetros obtenidos fueron $nf = 64$, ADAM como método de descenso por el gradiente, $wr = 0{,}0001$ y $\eta = 0{,}001$.

Con estos valores definidos se procede a entrenar la red durante 50 épocas. La

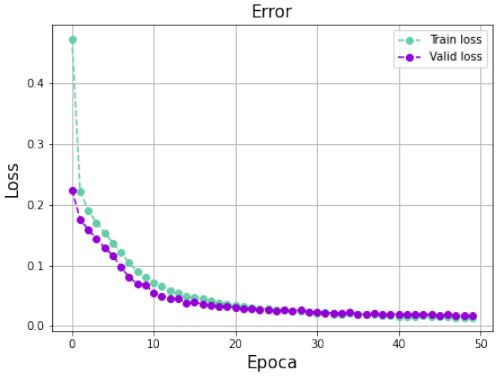


Figura 2.3: Función de costo de entrenamiento y validación durante 50 épocas del modelo de clasificación en el rango completo de ángulos de rotación con 4 clases.

evolución de la función de costo en función de las épocas a lo largo del entrenamiento se grafica en la Figura 2.3.

Se puede ver en el gráfico 2.3 que el valor del error en el conjunto de validación es menor que el calculado sobre el conjunto de entrenamiento y además no tiene una mejora significativa a partir de la época 30 por lo que se decidió hacer un entrenamiento con dicha cantidad épocas, agregar un loop nueva para lograr arreglar las curvas y hacer una comparación de su rendimiento. Para hacer dicha comparación se procede a ver, además del error mostrado en la Figura 2.4, si las métricas tienen una mejora significativa entre entrenarla con 50 o 30 épocas. Para ello se construyen las matrices de confusión sobre el conjunto de evaluación (Figura 2.5) con las cuales se calculan las métricas correspondientes (1.2.6) cuyos resultados se muestran en la Tabla 2.1. En este caso las matrices que se utilizan son las normalizadas sobre las condiciones verdades, osea sobre las filas.

Como se ve en la Tabla 2.1, las métricas obtenidas a partir del entrenamiento con 50 épocas no varían en más de 0,24 % de aquellas calculadas con los datos del entrenamiento con 30 por lo que, para ganar tiempo de computo, se estableció el tiempo de los entrenamientos en 30 épocas.

Se puede observar que con el nuevo loop los valores de la función de costo en el conjunto de entrenamiento quedan por debajo de aquellos evaluados sobre el conjunto de validación como es de esperarse.

Al ver los valores obtenidos de las métricas calculadas podemos concluir que el

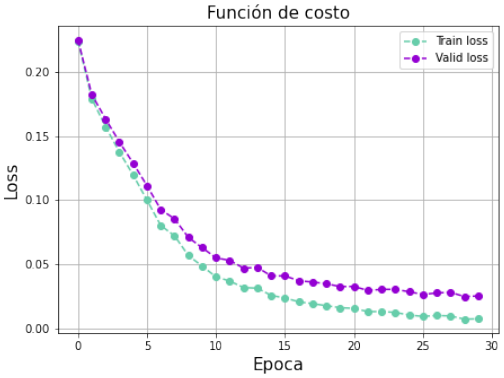
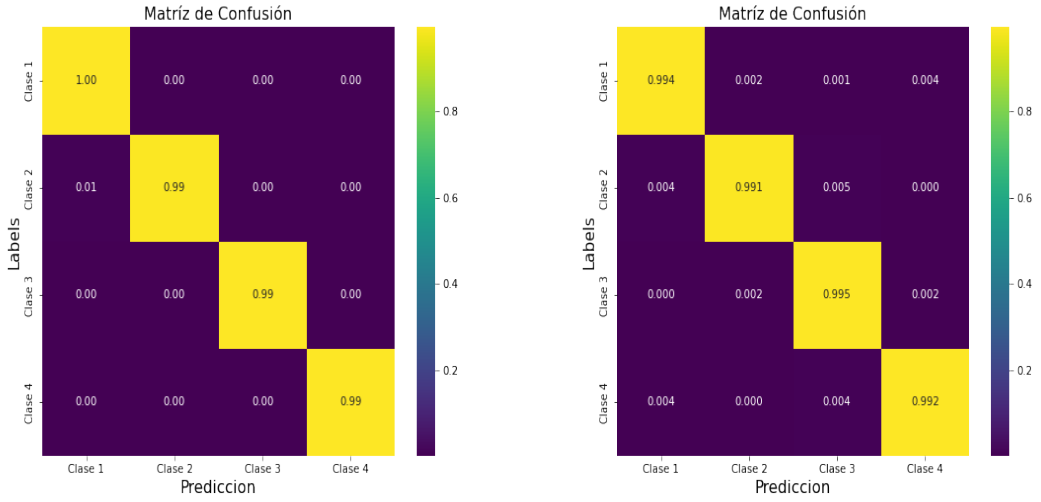


Figura 2.4: Función de costo de entrenamiento y validación durante 30 épocas del modelo de clasificación en el rango completo de ángulos de rotación con 4 clases.



(a) Entrenamiento con 50 épocas.

(b) Entrenamiento con 30 épocas.

Figura 2.5: Matrices de Confusión normalizadas evaluadas sobre el conjunto de testeo del modelo de clasificación en el rango completo de ángulos de rotación con 4 clases.

Métricas			
Métrica	Clase	Valor obtenido[%]	
		50 Épocas	30 Épocas
Exactitud		99.38	99.29
Precisión			
	1	99.05	99.11
	2	99.65	99.49
	3	99.40	99.18
	4	99.43	99.39
	MAP	99.38	99.29
Sensibilidad			
	1	99.64	99.40
	2	99.05	98.99
	3	99.48	99.50
	4	99.36	99.27
	MAR	99.38	99.29
MF1		99.38	99.29

Tabla 2.1: Métricas obtenidas de los entrenamientos con 50 y 30 épocas del modelo de clasificación en el rango completo de ángulos de rotación con 4 clases.

rendimiento de la red en la tarea de reconocer grupos de ángulos de rotación es muy buena, superando el 99 % en todos los casos.

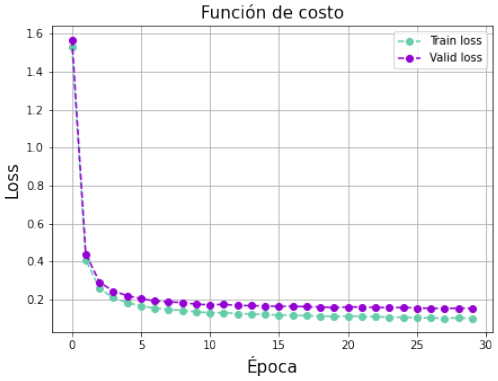


Figura 2.6: Función de costo de entrenamiento y validación durante 30 épocas del modelo de clasificación en el rango completo de ángulos de rotación con 360 clases.

2.1.2. Clasificación 360 clases

Luego de estos muy alentadores resultados, lo que intentamos a continuación es seguir utilizando la opción de predecir una clase particular pero donde cada una represente un solo ángulo. La red, dado este problema, debería ser capaz de producir a la salida un vector de dimensionalidad 360. Cada entrada de ese vector representara la probabilidad de que la clase sea la correcta. Para poder llevar a cabo esta parte se deberán modificar las etiquetas de los ejemplos, dejando la primera clase correspondiente a ángulos de rotación de 0 grados y así, sumando de a un grado llegamos hasta la última que corresponderá con 359°.

Para entrenar dicha red utilizaremos los mismos hiper-parámetros que se encontraron en la optimización con Optuna para el modelo anterior, es decir $nf = 64$, ADAM como método de descenso por el gradiente, $wr = 0{,}0001$ y $\eta = 0{,}001$. También se usan 30 épocas para entrenar. El único cambio es que se utiliza el conjunto de dígitos completo para crear las rotaciones.

Los resultados obtenidos de dicho entrenamiento se muestran en la Figura 2.6 donde se ve la evolución de la función de costo por época.

Después de 30 épocas, la red logra llegar a un valor estable del error en el conjunto de validación. Eso es bastante bueno, especialmente considerando que muchas de las muestras originales en el conjunto de datos MNIST no están orientadas siempre igual, es decir que pueden poseer variabilidad del catalogado "ángulo cero".

Una vez que finaliza el entrenamiento, podemos usar el modelo para predecir

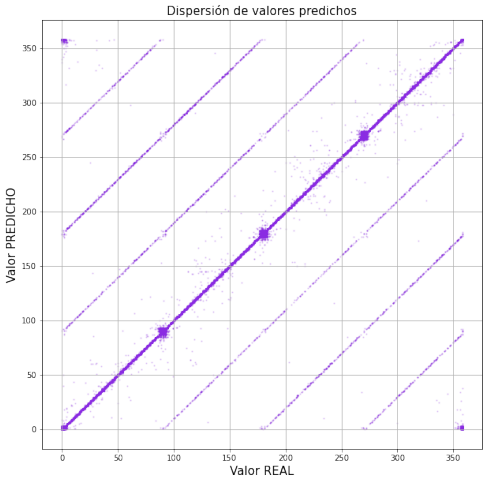


Figura 2.7: Gráfico de valores predichos en función de la etiqueta real del modelo de clasificación en el rango completo de ángulos de rotación con 360 clases.

el ángulo de rotación de cualquier imagen del conjunto de datos MNIST. Para ello utilizamos el conjunto de evaluación que hemos creado. En la Figura 2.7 podemos observar la dispersión de las predicciones de los ejemplos presentados a la red en función en función de la etiqueta real. Vemos que aparecen muchas rectas, todas con pendiente de valor unitario, pero donde sus ordenadas al origen difieren en 90° entre rectas contiguas. Si bien no se logra apreciar con facilidad en dicho gráfico, cuando calculamos la exactitud del modelo obtenemos un valor de 90,97 % con lo que podemos afirmar que la recta que más puntos posee es la central, que corresponde a los ejemplos bien etiquetados.Esto se debe a las simetrías de rotación ya comentadas de muchos de los dígitos.

2.2. Clasificación restringida

El siguiente paso para las redes clasificadoras fue restringir los ángulos de rotación de las imágenes. Para ello se volvieron a crear las rotaciones del conjunto de datos pero esta vez restringiendo el rango de ángulos de rotación en el intervalo $[-60, 60]$. Esta simplificación se debe a que los humanos tenemos la habilidad de procesar imágenes a un rango bastante acotado. Enmarcado en esta sección se realizaron dos

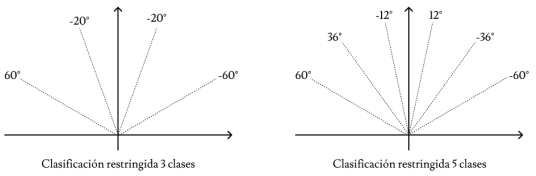


Figura 2.8: Gráfico de las diferentes etiquetas de clasificación para la red clasificadora restringida con 3 y 5 clases.

clasificaciones distintas dependiendo de la cantidad de categorías posibles a la salida. La primera consta de 3 clases de 40° cada una y la segunda de 5 clases de 24°. En ambas configuraciones dejamos las clases ordenadas simétricamente alrededor del 0°, es decir que las etiquetas se empiezan a catalogar ascendentemente, partiendo de la çlase 1çomo la correspondiente al primer intervalo, comenzando desde los $-60°$, de forma que la clase central corresponda a los intervalos de $[-20, 20]$ y $[-12, 12]$ respectivamente para 3 y 5 clases (Figura 2.8).

En esta sección se seguirán utilizando los hiper-parámetros optimizados para el problema de la sección anterior mediante el uso de la librería Optuna, el subconjunto de dígitos del MNIST (3, 4 y 7) antes descripto y 30 épocas.

2.2.1. Clasificación 3 clases

Como se mencionó recién, lo que se busca ahora es que la red aprenda a distinguir entre ángulos de rotación en un intervalo restringido de ángulos $[-60, 60]$ con 3 neuronas de salida pertenecientes a diferentes clases. Los resultados obtenidos de dichos entrenamientos se muestran en la Figura 2.9, donde nuevamente se grafican las curvas del error de entrenamiento y validación en función de la época.

Al igual que antes, de dicho entrenamiento se realizo la matriz de confusión, evaluando sobre el conjunto de evaluación, obteniendo los resultados que se muestran en la Figura 2.10.

Con estos valores se calculan las métricas asociadas a la red, dichos resultados se muestran en la Tabla 2.2.

Notamos como el desempeño es muy bueno, haciendo que todos los parámetros que reflejan el rendimiento de la red superen el 99,5 % en todos los casos.

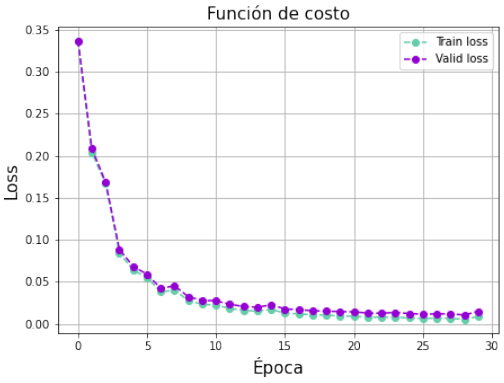


Figura 2.9: Evolución de los valores de la función de costo de entrenamiento y validación por época para el modelo de clasificación restringida en ángulos con 3 clases.

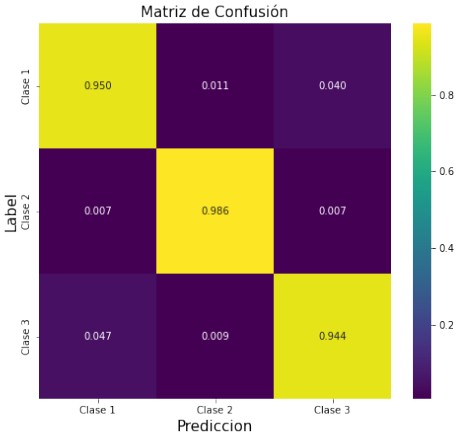


Figura 2.10: Matriz de confusión obtenida de la evaluación del modelo de clasificación restringida en ángulos con 3 clases.

Métrica	Clase	Valor obtenido [%]
Exactitud		99.66
Precisión		
	1	99.65
	2	99.60
	3	99.73
	MAP	99.66
Sensibilidad		
	1	99.75
	2	99.59
	3	99.63
	MAR	99.66
MF1		99.66

Tabla 2.2: Métricas obtenidas de los entrenamientos del modelo de clasificación restringida en ángulos con 3 clases.

2.2.2. Clasificación 5 clases

En esta sección mostramos los mismos resultados que los recién mostrados para 3 clases obtenidos del entrenamiento y evaluación de la red con 5 neuronas de salida. Podemos ver la evolución de la función de costo en la Figura 2.11, su matriz de confusión y sus métricas en la Figura 2.12 y en la Tabla 2.3 respectivamente.

Como era de esperarse, los resultado obtenidos con 3 clases es siempre mejor que los obtenidos con 5 clases, dado que los rangos en los cuales debe predecir y clasificar son más amplios. De todas formas los resultados con esta arquitectura siguen reflejando un gran desempeño. Esto puede deberse a que tanto los entrenamientos como las evaluaciones se están haciendo sobre un subconjunto del conjunto de datos completo, previamente elegido, que facilita el aprendizaje.

El siguiente paso, por lo tanto, fue evaluar el desempeño de la red entrenada con el subconjunto antes mencionado, pero testeando su poder de generalización con imágenes de dígitos distintos al del conjunto de entrenamiento. Esto se hizo solo para el caso en el cual los ángulos de rotación se clasificaron en 5 categorías. Para ello se creo un nuevo conjunto de evaluación con el resto de las imágenes MNIST (es decir el conjunto de las imágenes de dígitos $0, 1, 2, 5, 6, 8$ y 9), generando nuevamente 20 imágenes rotadas de cada una, como se había hecho en el caso anterior. Algunos

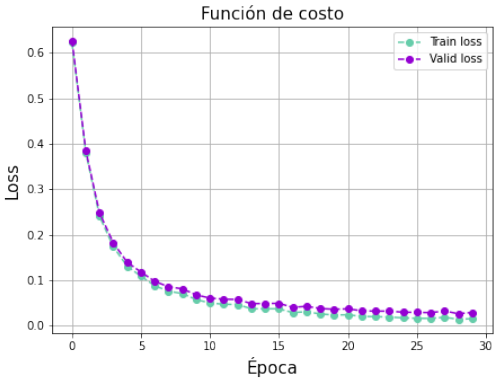


Figura 2.11: Evolución de los valores de la función de costo de entrenamiento y validación por época para el modelo de clasificación restringida en ángulos con 5 clases.

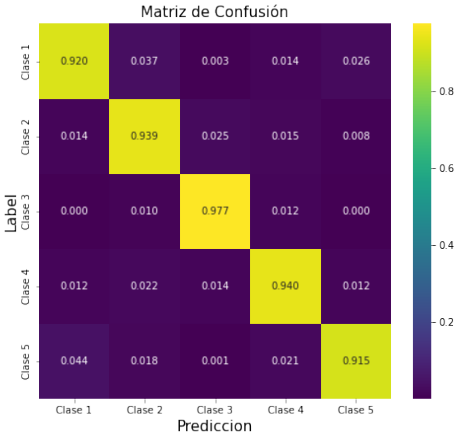


Figura 2.12: Matriz de confusión obtenida de la evaluación del modelo de clasificación restringida en ángulos con 5 clases.

Métrica	Clase	Valor obtenido [%]
Exactitud		99.01
Precisión		
	1	99.37
	2	99.03
	3	99.20
	4	99.63
	5	99.84
	MAP	99.02
Sensibilidad		
	1	98.60
	2	99.08
	3	99.02
	4	99.33
	5	99.02
	MAR	99.01
MF1		99.01

Tabla 2.3: Métricas obtenidas de los entrenamientos del modelo de clasificación restringida en ángulos con 5 clases.

ejemplos con su respectiva etiqueta se muestran en la imagen 2.13.

Con este nuevo conjunto se evaluó la red para ver si es capaz de identificar, nuevamente, categorías de rotaciones sobre datos con los cuales no fue entrenada. Con esto se genera la correspondiente matriz de confusión y se calculan las métricas que se muestran en la Figura 2.14 y la Tabla 2.4 respectivamente.

Lo primero que se puede decir es que, como era de esperarse, el rendimiento de la red disminuye respecto al anterior. De todas formas las valores calculados siguen estando por encima del 88 % para todas las métricas. Otro punto a destacar es que, tanto en la precisión como en la sensibilidad, el mejor desempeño de la red se da en la clase central (que contiene al ángulo 0), superando notablemente el valor de las demás clases.

Finalmente, alentados por el rendimiento obtenido en todos los casos anteriores, lo que se hizo fue un entrenamiento y evaluación de la red restringida con 5 categorías pero entrenada con todos los diez dígitos del conjunto MNIST. Dado que los ángulos de rotación estaban contenidos en el intervalo $[-60, 60]$, el único número que podría confundir a la red, dada su simetría rotacional, es el cero, ya que en

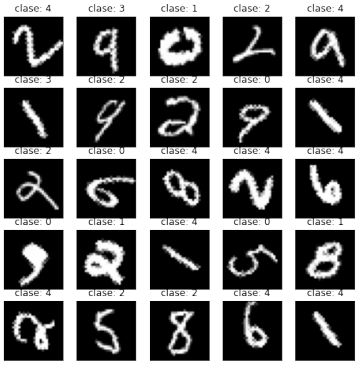


Figura 2.13: Ejemplo de las nuevas imágenes del conjunto de testeo.

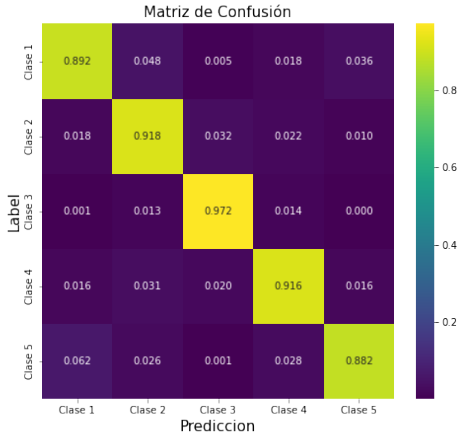


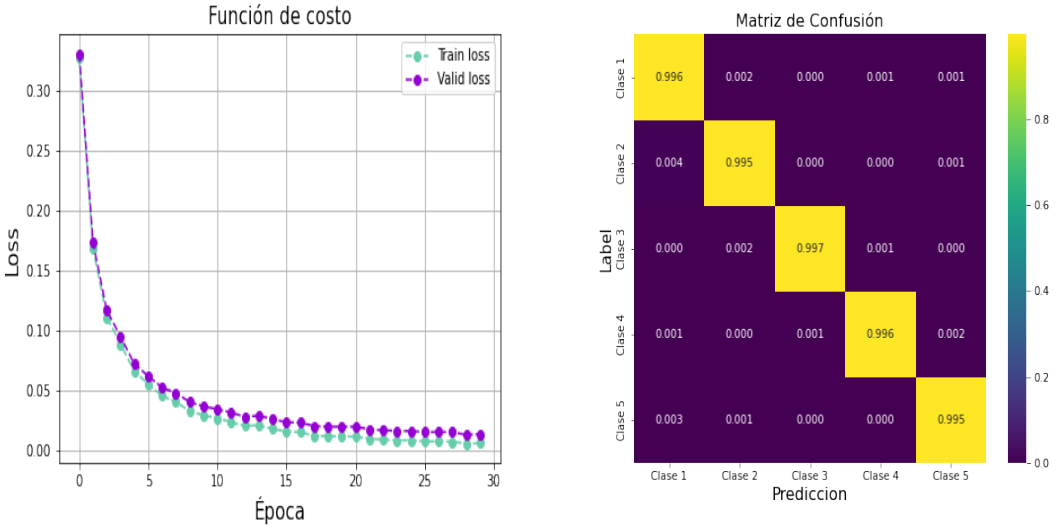
Figura 2.14: Matriz de confusión normalizada del entrenamientos del modelo de clasificación restringida en ángulos con 5 clases evaluada sobre conjunto de datos de dígitos con los que no fue entrenada.

Métrica	Clase	Valor obtenido [%]
Exactitud		91.62
Precisión		
	1	90.13
	2	88.60
	3	94.44
	4	91.76
	5	93.27
	MAP	91.64
Sensibilidad		
	1	89.01
	2	91.82
	3	97.08
	4	91.81
	5	88.35
	MAR	91.61
MF1		91.63

Tabla 2.4: Métricas obtenidas del entrenamientos del modelo de clasificación restringida en ángulos con 5 clases sobre conjunto de datos con dígitos con los que no fue entrenada.

dicho intervalo el resto de las simetrías antes mencionadas pierden validez. De todas formas incluimos el cero como parte del entrenamiento y evaluación del modelo. Los resultados obtenidos con este entrenamiento se muestran en las figuras 2.15a y 2.15b y en la Tabla 2.5.

Si comparamos el problema de clasificar ángulos previamente etiquetados en cinco categorías en el intervalo $[-60, 60]$ cuando entrenamos en el conjunto restringido de dígitos (3, 4 y 7) con el caso en el cual entrenamos sobre los diez dígitos, vemos que este último caso tiene mejor resultado. Esto puede parecer confuso, pues en el caso restringido las simetrías son menos importantes. Sin embargo al entrenar la red en un conjunto mucho más amplio de imágenes de entrenamiento, hemos permitido que la red aprenda de manera más confiable a resolver el problema. Calculando las métricas con los valores de ejemplos predichos es que obtenemos los resultados mostrados. Nuevamente todos los valores calculados superan el 99 %.



(a) (b)

Figura 2.15: (a) Evolución de los valores de la función de costo de entrenamiento y validación por época para el modelo de clasificación restringida en ángulos con 5 clases. (b) Matriz de confusión normalizada para el modelo de clasificación restringida en ángulos con 5 clases evaluada sobre el conjunto de entrenamiento con todos los números de MNIST.

Métrica	Clase	Valor obtenido [%]
Exactitud		99.56
Precisión		
	1	99.09
	2	99.42
	3	99.87
	4	99.82
	5	99.60
	MAP	99.56
Sensibilidad		
	1	99.56
	2	99.50
	3	99.69
	4	99.51
	5	99.53
	MAR	99.56
MF1		99.56

Tabla 2.5: Métricas obtenidas del entrenamientos para el modelo de clasificación restringida en ángulos con 5 clases sobre conjunto de datos usando todos los números de MNIST.

Capítulo 3

Regresión

En esta segunda parte del trabajo se implementan redes neuronales para resolver el problema de la rotación de imágenes en forma regresional. Es decir, la salida esperada de la red es un único valor que representa el "ángulo de rotación de la imagen de entrada predicho" por la red. Como también se trata de redes convolucionales, las entradas de la red vuelven a ser transformaciones del conjunto MNIST, equivalentes a las descriptas en el capitulo anterior. La diferencia es que para esta tarea la etiqueta o salida deseada corresponde al ángulo de rotación y no al intervalo en el cual se encuentra dicho ángulo. Nuevamente se realizan 20 rotaciones de las imágenes de entrada con los ángulos de rotación acotados en el intervalo $[-60, 60]$.

En esta parte del trabajo no se fuerza a la red a hacer una cantidad de rotaciones por intervalo sino que se generan números enteros aleatorios en el intervalo deseado y se los pasa como parámetro de la función de rotación (mismo numero que queda como etiqueta del ejemplo). Para verificar que los datos creados (las rotaciones) están bien distribuidos en el rango de valores se realiza el correspondiente histograma, el cual no se presenta.

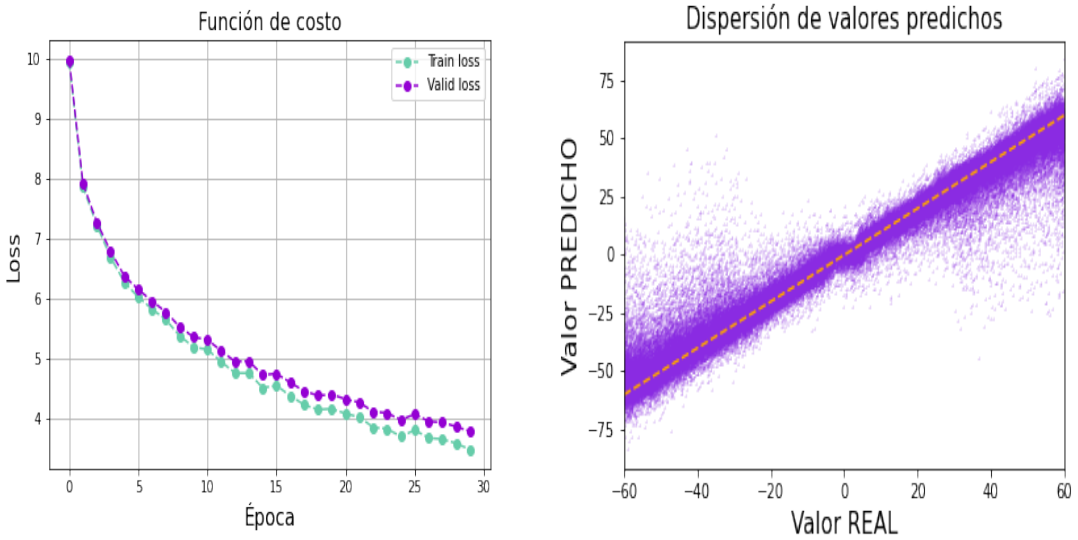
Se utilizan dos arquitecturas diferentes. La primera es la misma que se utiliza en la red clasificadora, es decir una red con arquitectura de dos capas convolucionales, por lo que la denotaremos como arquitectura "2C". La segunda consta de tres capas convolucionales sucesivas "3C", todas ellas con función de activación ReLU, de 8, 16 y 32 filtros de tamaño 3×3 respectivamente. Todas tienen un valor de paso de

2 y mientras las dos primeras capas cuentan con un rellenado de 1, la última no lo utiliza. Con estos hiper-parámetros la salida de cada capa genera una reducción de los píxeles a la mitad, llegando a una imagen de tamaño mínimo de 3×3. Luego de eso tenemos 3 capas lineales con 128, 64 y 1 neurona respectivamente. Esta última es la capa de salida, que esperamos prediga correctamente el ángulo de rotación. Las funciones de costo utilizadas en esta parte del trabajo son tanto la MSE como la MAE. Para cada una de las arquitecturas se compara el aprendizaje de la red cuando se usa cada una de las mencionadas funciones de costo.

Cabe destacar que la decisión de usar estas funciones de costo radica en que los valores de ángulos de rotación están restringidos al intervalo seleccionado pues si se trabajara con la vuelta completa de ángulos estas funciones presentarían una complicación en las proximidades del ángulo 0°. Para ejemplificar este problema, imaginemos que la imagen esta rotada 5° pero la red predice que el ángulo de rotación es de 355°, entonces, si utilizamos como medida la función de costo MAE, el error sería de 350°. Pero eso no tiene sentido para nosotros pues sabemos que la diferencia entre dichos ángulos en verdad es solo de 10°. Exactamente el mismo error que entre 5° y 15°.

Como ya se ha comprobado que la red es capaz de aprender utilizando todo el conjunto de datos, sin necesidad de restringirlo a aquellos números que no presentan problemas por su simetría, para el entrenamiento de estas redes se utiliza el conjunto de dígitos completo de MNIST, con las modificaciones mencionadas de rotaciones y etiquetas antes descriptas.

Como en esta parte de la tesina se trabaja con dos arquitecturas distintas y en cada una de ellas se utilizan dos funciones de costo, a continuación se muestran cuatro entrenamientos correspondientes a las arquitecturas de dos capas (2C) y tres capas (3C) con funciones de costo MAE (o L1) y MSE. Todos los entrenamientos se realizan durante 30 épocas, con la configuración de hiper-parámetros con $\eta = 0{,}001$, $wr = 0{,}0001$. y utilizando el método de descenso por el gradiente ADAM.



(a) (b)

Figura 3.1: Arquitectura 2C con función de costo MAE. (a) Evolución de la función de costo en el conjunto de entrenamiento y validación por época. (b) Gráfico de valores predichos vs valores reales.

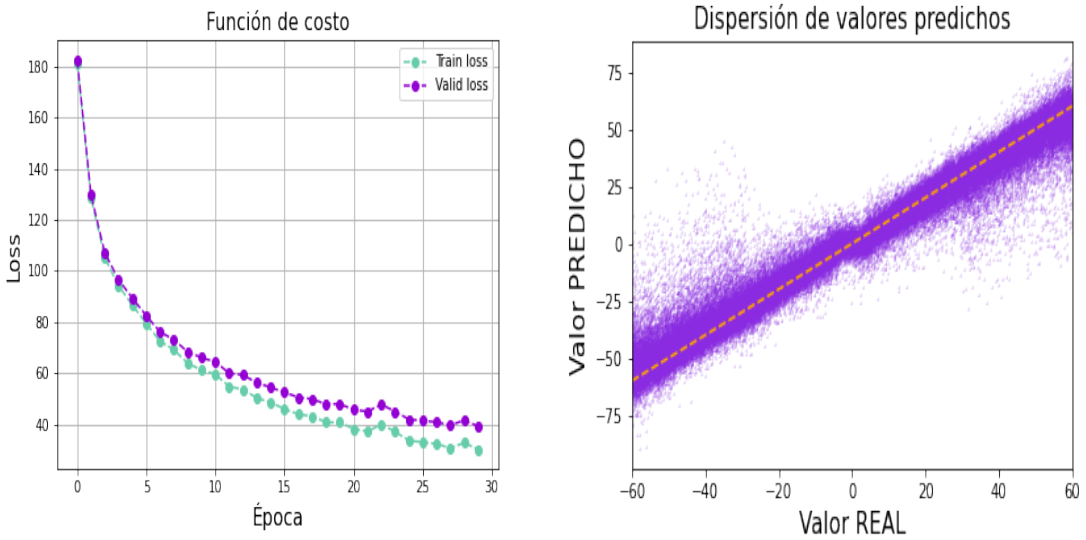
3.1. Arquitectura de dos capas

3.1.1. 2C MAE

En la Figura 3.1 se muestran los resultados obtenidos del entrenamiento de la red 2C con función de costo MAE. Mientras que en la Figura 3.1a se muestra la evolución de la función de costo en los conjuntos de entrenamiento y validación en función de las épocas, en la 3.1b se muestra un gráfico del valor predicho por la red cuando se le pasa el conjunto de testeo vs. el valor real de la etiqueta de dicho ejemplo junto con la función lineal identidad. Idealmente, cuanto mejor aprende la red, más cerca de la función identidad deberían caer los pares ordenados dados por la predicción y el valor real. Dado que siempre hay errores vemos una dispersión de los puntos.

3.1.2. 2C MSE

El segundo entrenamiento fue realizado con la misma arquitectura, con los mismos conjuntos de datos, cambiando la función de costo por la MSE. En la Figura



(a) (b)

Figura 3.2: Arquitectura 2C con función de costo MSE. (a) Evolución de la función de costo en el conjunto de entrenamiento y testeo en cada época. (b) Gráfico de valores predichos vs valores reales.

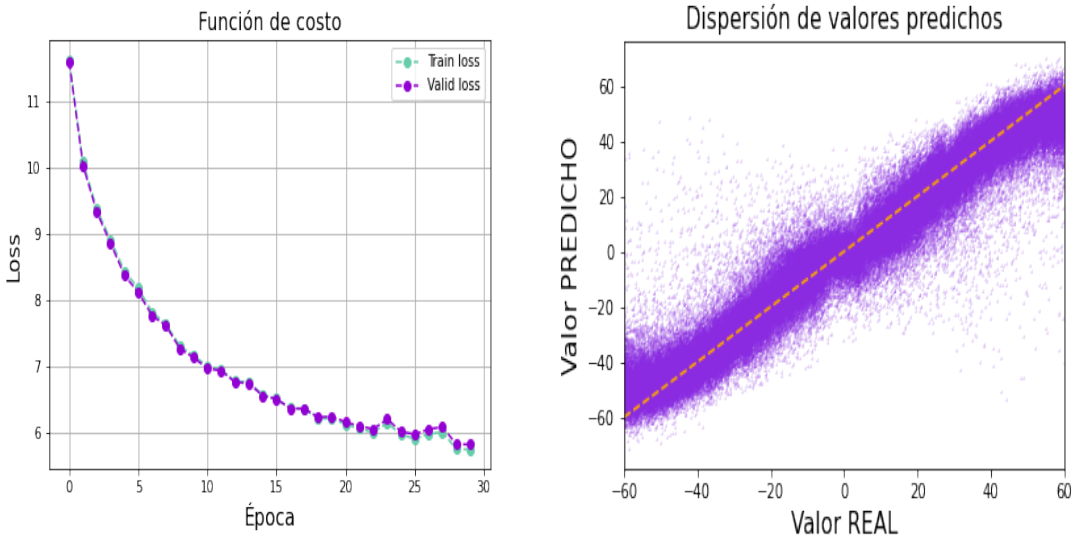
3.2 se muestran los resultados obtenidos con la configuración recién mencionada.

3.2. Arquitectura de tres capas

A continuación se cambia la arquitectura de la red por una más profunda, con tres capas convolucionales (3C), pero con menos filtros en cada capa. También se realizaron dos entrenamientos diferentes cambiando la función de costo en cada caso. Aquí se muestran los resultados obtenidos con ambas funciones de costo utilizadas.

3.2.1. 3C MAE

En esta parte se muestra el entrenamiento cuando se utilizó la función de costo MAE. Nuevamente la evolución de la función de costo utilizada para los conjuntos de entrenamiento y validación y el grafico de las predicciones del modelo en función de los valores esperados se ven en la Figura 3.3.



(a) (b)

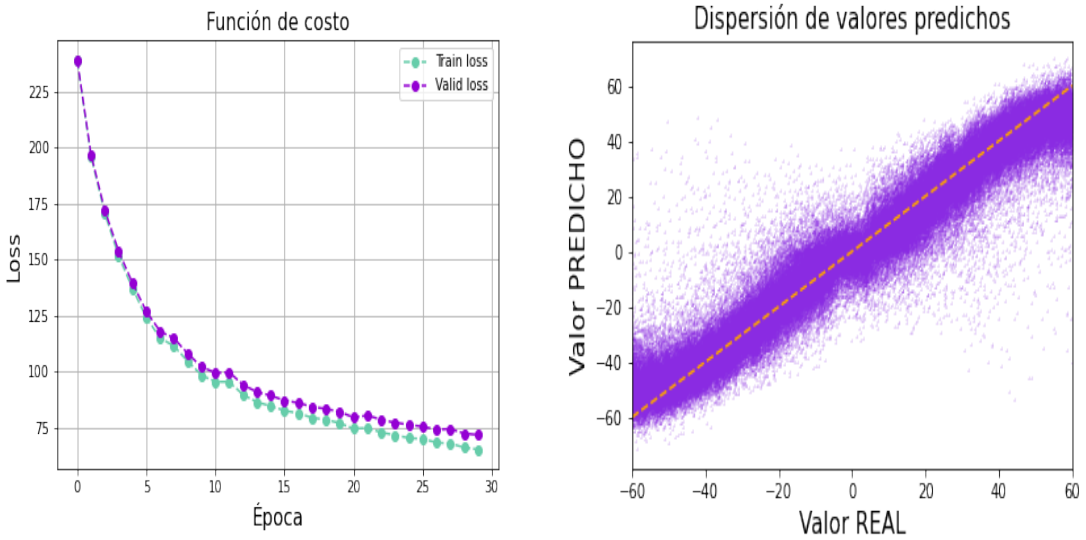
Figura 3.3: Arquitectura 3C con función de costo MAE. (a) Evolución de la función de costo en el conjunto de entrenamiento y testeo en cada época. (b) Gráfico de valores predichos vs valores reales.

3.2.2. 3C MSE

Cuando cambiamos la función de costo por la MSE la Figura 3.4 muestra los resultados obtenidos de la evolución de la función de costo por época y la dispersión de las predicciones del modelo.

3.3. Métodos de comparación

A fin de poder comparar los resultados obtenidos de las distintas arquitecturas entrenadas con las diferentes funciones de costo se utilizaran dos métodos de análisis. El primero consiste en realizar el ajuste lineal de los datos predichos por la red luego de ser entrenada en función del valor real del ángulo de rotación puesto como etiqueta, con el fin de ver cual se ajusta más a la función lineal Identidad. El segundo discretiza el espacio de salida de las redes regresionales en 8 clases, como si se tratara de una clasificación, obteniendo clases de 15° cada una, con la cual se puede realizar la matriz de confusión para calcular así algunas de las métricas de medición de desempeño de la red.



(a) (b)

Figura 3.4: Arquitectura 3C con función de costo MSE. (a) Evolución de la función de costo en el conjunto de entrenamiento y testeo en cada época. (b) Gráfico de valores predichos vs valores reales.

3.3.1. Ajustes lineales

A continuación se muestran los gráficos con la dispersión de datos del valor predicho vs. el valor real con sus correspondientes ajustes lineales realizados en **"Origin"**. En la leyenda de cada gráfico se pueden ver las pendientes y las ordenadas al origen de los ajustes correspondientes (Figura 3.5).

Como se puede ver en los gráficos 3.5a y 3.5b es notable como en la proximidad del ángulo 0° (aproximadamente ±5°) las predicciones del modelo son todas próximas a cero. Esto puede deberse a que las imágenes originales del conjunto MNIST tienen cierta varianza, donde ejemplos del mismo número se presentan en orientaciones levemente distintas. Esto hace que puedan existir dos imágenes con etiquetas de ángulo distintas que posean el mismo mapa de características o uno muy similar, lo que puede confundir a la red y entonces tiende a catalogar todas esas características como rotación nula.

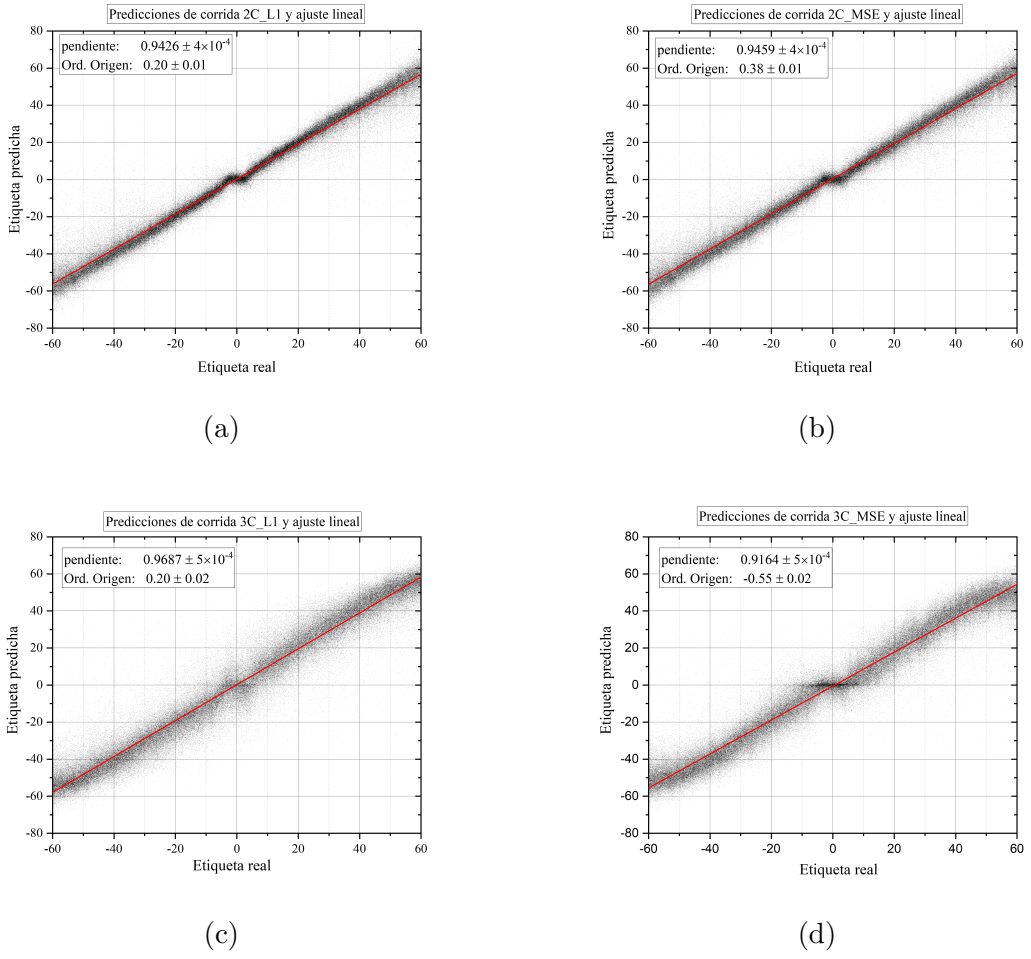
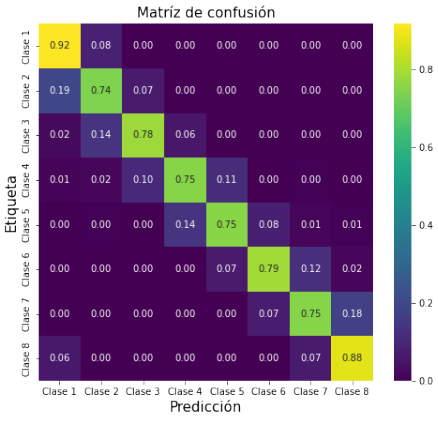
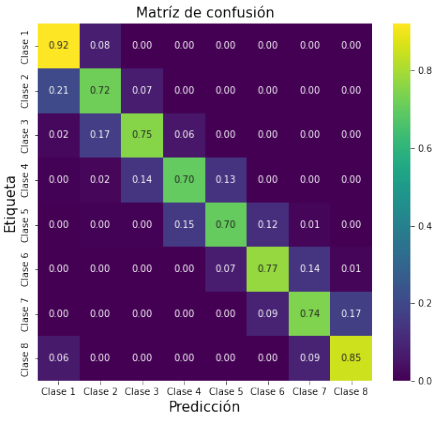


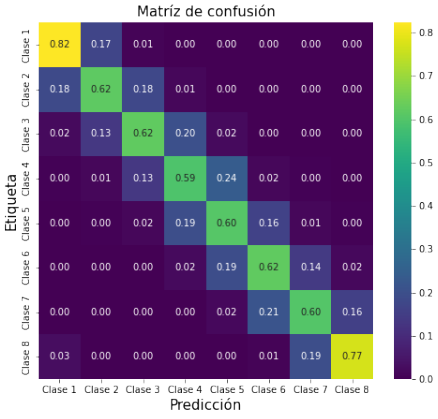
Figura 3.5: Gráficos de la distribución de puntos predicción vs real con sus respectivos ajustes lineales.



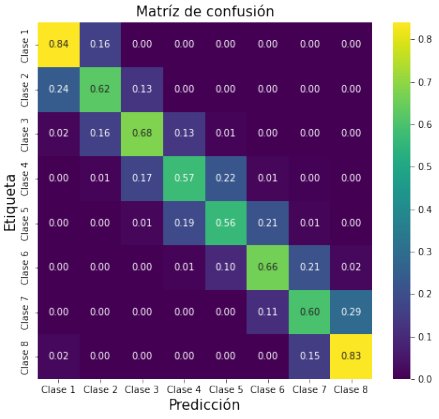
(a) Matriz de confusión 2C L1



(b) Matriz de confusión 2C MSE



(c) Matriz de confusión 3C L1



(d) Matriz de confusión 3C MSE

Figura 3.6: Matrices de confusión normalizadas para los modelos regresionales discretizados en 8 clases.

3.3.2. Discretización de valores de salida

En la Figura 3.6 podemos ver las cuatro matrices de confusión que se obtienen al discretizar los modelos de regresión. Nuevamente las matrices están normalizadas sobre las filas. Con estos datos se calcula, para cada una de las distitnas redes entrenadas con distintas funciones de costo, su exactitud, su precisión y sensibilidad para finalmente poder calcular la F1-score. Los valores obtenidos de dichas métricas se muestran en la Tabla 3.1.

Podemos ver como en las matrices de confusión sigue siendo la diagonal principal la más poblada, representando esto el buen desempeño de las redes. De todas for-

Métricas				
	2C L1	2C MSE	3C L1	3C MSE
Exactitud	78.87	76.32	65.60	65.80
MAP	78.88	76.33	65.59	65.80
MAR	79.45	77.05	65.73	67.14
F1-Score	79.16	76.69	65.66	66.46

Tabla 3.1: Métricas calculadas a partir de las matrices de confusión para los modelos de regresión discretizados en 8 clases.

Métricas				
	2C L1	2C MSE	3C L1	3C MSE
Exactitud	88.52	86.99	79.28	81.39
MAP	88.53	86.99	79.27	81.38
MAR	88.78	87.32	79.19	81.85
F1-Score	88.66	87.15	79.23	81.61

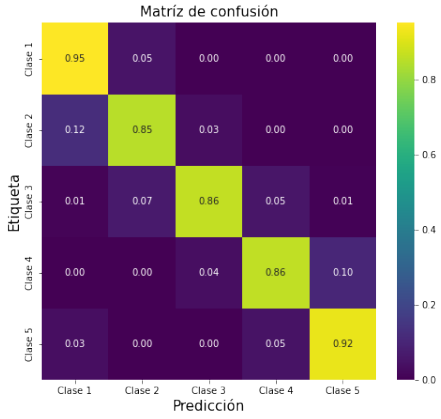
Tabla 3.2: Métricas calculadas a partir de las matrices de confusión para los modelos de regresión discretizados en 5 clases.

mas, notamos su gran disminución de rendimiento en comparación con los diferentes modelos de clasificación, aún cuando comparamos con el modelo de 360 clases.

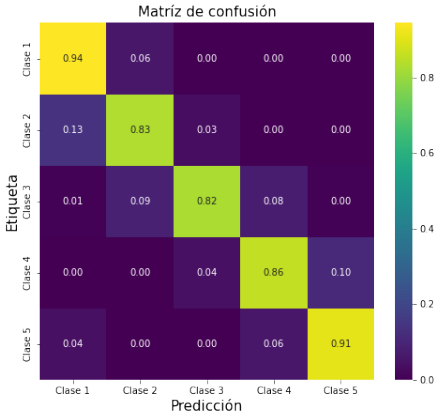
La decisión de hacer 8 clases de 15° cada una surgió del análisis de los gráficos de los valores predichos vs los reales. Se pensó que con esta cantidad de clases iba a ser suficiente para obtener buenas métricas. Al ver que las métricas disminuían significativamente comparados con la clasificación en 5 clases se decidió discretizar en esa cantidad de clases para hacer una comparación mas fiel con el modelo de clasificación. Al hacerlo se obtuvieron las matrices de confusión mostradas en 3.7 y sus respectivas métricas 3.2.

Se puede ver que mejoro bastante respecto del analisis con 8 clases pero de todas formas se siguen obteniendo rendimientos menores que los obtenidos con clasificación.

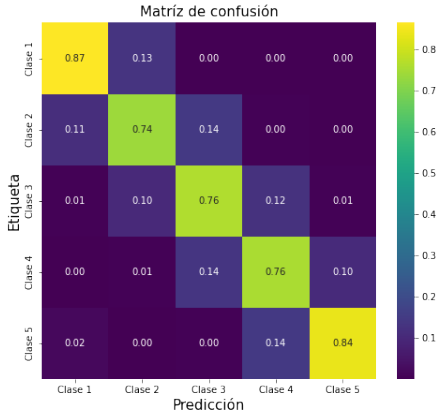
En modelos de regresión encontramos valores más altos de la sensibilidad que de precisión en todos las arquitecturas presentadas, valor que podemos interpretar como que hay mejor desempeño al contabilizar cuantos, del total de ejemplos catalogadas de cierta clase, se predijeron como tal que cuando contamos cuantos de los ejemplos



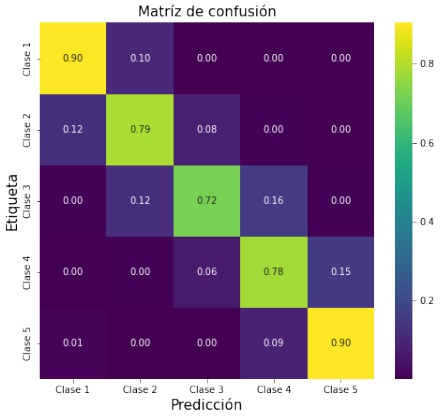
(a) Matriz de confusión 2C L1



(b) Matriz de confusión 2C MSE



(c) Matriz de confusión 3C L1



(d) Matriz de confusión 3C MSE

Figura 3.7: Matrices de confusión normalizadas para los modelos regresionales discretizados en 5 clases.

catalogados de cierta clase son efectivamente de esa clase.

Observemos que si bien la pendiente del ajuste lineal que más se aproxima a 1 es la de la arquitectura 3C con MAE, en la Tabla 3.1 podemos observar que la arquitectura con mejores rendimientos es la que consta de 2C con MAE. Esto puede deberse a que las predicciones de la 3C con MSE están más dispersas, logrando una mejor pendiente pero peores métricas.

Capítulo 4

Auto-Encoder Convolucional

Finalmente se realiza una red AEC. Como se ha dicho en el capitulo 1, los AEC son redes feed-forward, por lo que pueden usarse en su diseño las mismas unidades neuronales, funciones de activación y de costo que ya presentamos. La diferencia radica en que la salida esperada de este tipo de redes tiene la misma estructura que la entrada. En nuestro caso la entrada y la salida del AEC son imágenes bidimensionales de 28×28 píxeles. El conjunto de datos utilizados nuevamente es el MNIST completo. En esta parte del trabajo se espera poder entrenar a la red para que, dada una imagen de entrada previamente rotada como en las secciones anteriores, esta sea capaz de replicarla a la salida pero de forma "orientada" o "derecha". Para lograrlo se desarrolla un AEC con aprendizaje supervisado donde las entradas son las mismas que se le pasan a la red regresional pero, esta vez, la salida esperada de la red para cada ejemplo es la imagen "original" del conjunto MNIST, a partir de la cual se realizó la rotación. Así, tanto la entrada de la red como la salida, son datos estructurados.

A continuación explicamos la arquitectura utilizada que, como ya se menciono, consta de dos partes principales, el encoder y el decoder.

El encoder consta de tres capas convolucionales con filtros de tamaño 3×3 y función de activación ReLU. En las dos primeras los valores de stride y padding son 1 dejando, de esta forma, el tamaño de las salidas de dichas capas igual al tamaño de la imagen de entrada. En la tercera el valor de stride es cambiado a 2, dejando el

padding igual a 1 de forma tal que en dicha capa el tamaño de salida se reduzca a la mitad en alto y ancho, dejando imágenes de salida de tamaño 14×14. La cantidad de filtros en cada capa es de 16, 32 y 64 respectivamente. Estas tres capas son seguidas de dos capas lineales totalmente conectadas con función de activación ReLU, donde la primera contiene la misma cantidad de unidades de entrada que la salida de la tercera capa convolucional, es decir $14 \times 14 \times 64$, y la salida de la misma es de 1024 neuronas y la segunda solo posee 128 neuronas de salida.

La parte de la descompresión de datos, es decir el decoder, recorre el camino exactamente inverso al hecho en la codificación. Comienza tomando como entrada la última capa lineal de 128 neuronas del encoder y llevandola a 1024, seguida de la capa lineal de salida de $14 \times 14 \times 64$ neuronas. A continuación se realiza una reestructuración de las neuronas para volver a la estructura de 64 imágenes de dimensión 14×14 a través de una capa "unflatten". Finalmente se aplican las 3 capas convolucionales transpuestas "Transposed Conv2d" con filtros de tamaño 3×3 en orden inverso al encoder, es decir, primero la de 64 filtros de entrada; cuyos parámetros internos devuelven a la imagen al tamaño original de 28×28 (que las capas siguientes dejaran fijo) y salida de 32 filtros, luego la que lleva de 32 a 16 filtros y finalmente la capa donde entran estos 16 filtros y cuya salida final es una única imagen, a la que llamaremos la "reconstrucción". Todas las capas utilizan función de activación ReLU. La función de costo que usaremos es la MSE.

El entrenamiento de la red se realiza durante 30 épocas, con los siguientes valores de los hiper-parámetros: $\eta = 0{,}001$, $wr = 0{,}0001$ y con ADAM como método de descenso por el gradiente.

En la Figura 4.1 se muestra el gráfico de la evolución del valor de la función de costo tanto del conjunto de entrenamiento como del de validación a través de las 30 épocas. Observando las reconstrucciones obtenidas de dicho entrenamiento (Figura 4.2) podemos ver que el auto-encoder aprendió a realizar la tarea, reconstruyendo el dígito correcto de la imagen de entrada y orientándola de forma "derecha", aunque con ciertas imperfecciones. De todas formas, a pesar de que el modelo es simple, su desempeño es bastante bueno.

Como sabemos, estos métodos son muy poderosos y buenos para hacer predic-

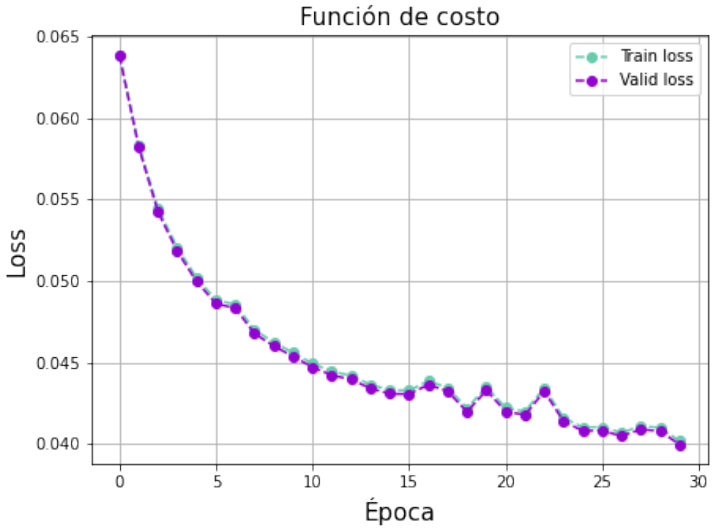


Figura 4.1: Evolución de la función de costo en el conjunto de entrenamiento y validación.

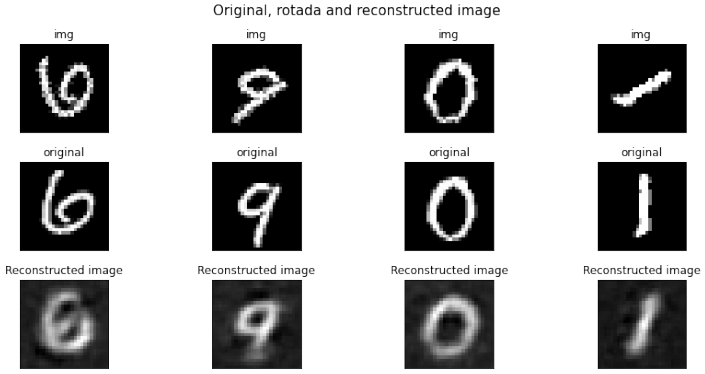


Figura 4.2: Cada columna contiene ejemplos de la imagen original (etiqueta), la imagen rotada (entrada) y su reconstrucción (salida) de diferentes dígitos.

ciones, pero al mismo tiempo, son difíciles de interpretar. Por este motivo se suelen denominar modelos de *caja negra*. Sabemos que las redes neuronales convolucionales aprenden de las entradas a través de filtros, cada uno de los cuales identifica ciertas características abstractas a partir de píxeles de imágenes y crea los mencionados "mapas de caracterísiticas". Por este motivo, a continuación nos enfocamos en ver cómo estas redes aprenden dichas características. Esto se hace mediante la visualización de las funciones aprendidas en cada capa. Podemos visualizar los filtros aprendidos, utilizados por la red en cada capa convolucional, que contienen las características extraídas de la capa anterior (figuras 4.3, 4.4 y 4.5). Es decir, a con-

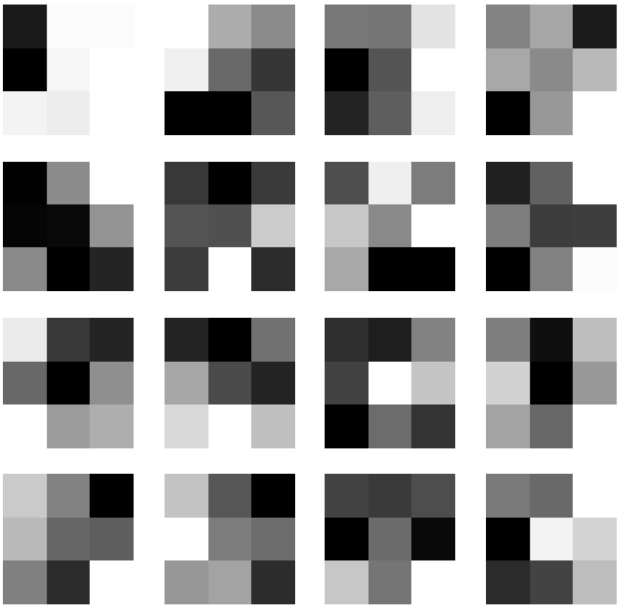


Figura 4.3: Filtros de la primera capa convolucional del AEC.

tinuación se grafican en escala de grises los pesos asociados a los diferentes filtros aprendidos en cada capa convolucional.

Además podemos observar el mapa de características, también llamado *mapa de activaciones*, que se obtiene al aplicar la operación de convolución a los datos de entrada mediante cada filtro.

Del conjunto de datos de entrenamiento, tomamos una imagen que representa el dígito 6 y otra que representa al dígito 1. De dichas entradas se visualizan los mapas de características obtenidos para cada imagen en la última capa convolucional, es decir la salida de la tercera capa, luego de aplicarle a su entrada los filtros mostrados en la Figura 4.5. Notemos que estas imágenes son del tamaño 14×14 pues así lo configuramos en la arquitectura de la red.

Vemos que hay muchos filtros que tienen un mapa de características completamente negro. Esto puede interpretarse como que la característica que reconoce dicho filtro no esta presente en la imagen que se le pasa como entrada. Lo raro es que en ambos dígitos son los mismos filtros los negros, es decir que dichos filtros no

Figura 4.4: Filtros de la segunda capa convolucional del AEC.

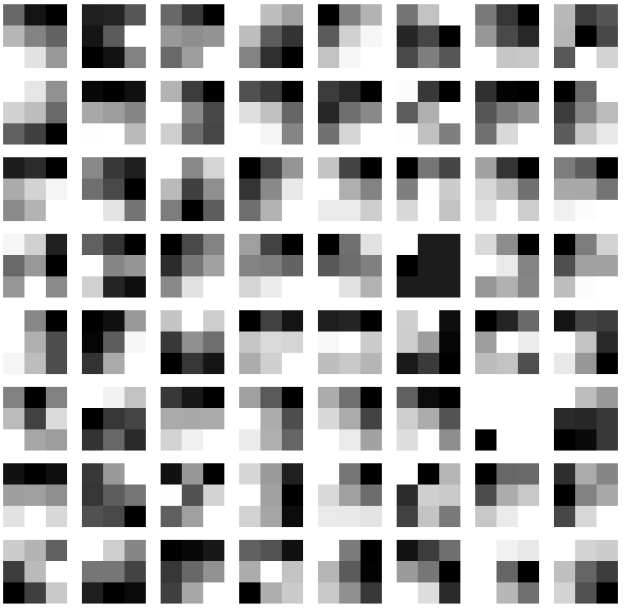
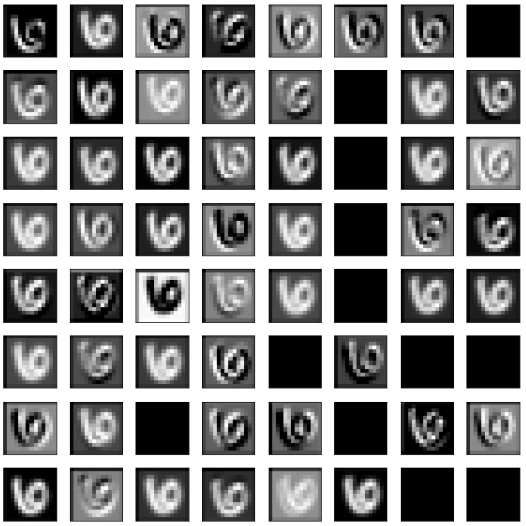
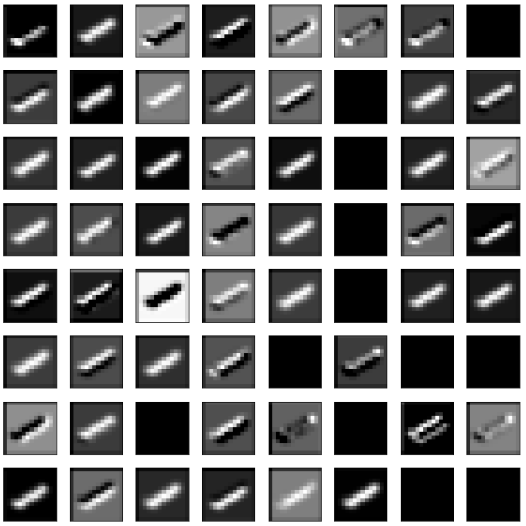


Figura 4.5: Filtros de la tercera capa convolucional del AEC.



(a) Mapa de activaciones obtenidos en la ultima capa convolucional con el dígito 6.



(b) Mapa de activaciones obtenidos en la ultima capa convolucional con el dígito 1.

Figura 4.6

están aprendiendo nada que sea de utilidad a la hora de aprender la representación intermedia del conjunto de datos.

Capítulo 5

Conclusiones

En este trabajo final hemos abordado la tarea de crear diferentes redes neuronales, con distintas arquitecturas, para reconocer ángulos de rotación en imágenes. La idea de este trabajo surge a raíz de trabajos anteriores, realizados por Dras. Carolina Tauro y Carolina Daza, en los que se estudia la forma de procesar información en la corteza visual del cerebro de mamíferos, también llamada V1. Dicha área presenta respuestas a estímulos visuales con determinada orientación que forman patrones observables de actividad sincronizada llamados "mapas de preferencia orientacional". Estos estudios modelan dicha actividad a través de redes neuronales. A diferencia de esos trabajos la idea de esta tesina fue diseñar redes neuronales convolucionales para su modelado. Si bien sabemos que existen ya redes que son capaces de realizar la tarea que estamos atacando, como la introducida en Krizhevsky et al. (2017) (AlexNet), en este trabajo utilizamos arquitecturas simples, de no más de 3 capas de profundidad.

Una de las dificultades al trabajar con redes neuronales es que necesitan ser entrenadas en un conjunto de datos lo suficientemente grande y variado como para poder aprender de la experiencia y detectar características comunes. Es por esto que en este trabajo hemos usado la técnica de aumento de datos, utilizando el conjunto MNIST y creamos nuestras propias imágenes rotadas con etiquetas correspondientes a la tarea que queríamos realizar.

La primer tarea fue de clasificación en pocas clases que abarcaban distintos

rangos de ángulos de rotación. Los resultados obteniendo en todas las arquitecturas y entrenamientos realizados tuvieron un muy buen desempeño.

En las redes entrenadas para clasificar en 3, 4 y 5 intervalos de ángulos de rotación se obtuvieron siempre desempeños que superaban los 99 puntos porcentuales en la evaluación de todas las métricas seleccionadas. Cabe destacar que si bien todas superaban ese valor, los mejores rendimientos se obtuvieron cuando entrenamos con el total de los dígitos del conjunto de datos. Esto puede deberse a que al entrenar la red en un conjunto mucho más amplio de imágenes de entrenamiento, se permitió que la red aprenda de manera más confiable. Como era de esperarse el rendimiento de la red cuando se evaluó sobre un conjunto de testeo con dígitos con los que no había sido entrenada disminuyó respecto a los recién mencionados. De todas formas se obtuvieron métricas que rondaban los 90 puntos porcentuales, lo cual sigue siendo un rendimiento sobresaliente.

En el caso particular de clasificación en 360 categorías (una por grado), no hemos calculado la matriz de confusión por la complejidad que representa. Sin embargo, a pesar de la dificultad de la tarea, observamos que la exactitud no ha decrecido demasiado. En algún sentido podemos pensar a esta clasificación como un problema equivalente a la regresión, y cuando se compara con regresión vemos que la clasificación en 360 categorías es mucho mejor. Vale destacar que esta capacidad de discernir en tantas categorías supera ampliamente las capacidades desarrolladas por los mamíferos a lo largo de la evolución.

La siguiente tarea fue tratar de predecir el ángulo de rotación, es decir una tarea de regresión. En esta parte comparamos dos arquitecturas que diferían en profundidad y cantidad de filtros por capa, además de utilizar dos funciones de costo diferentes. Estas arquitecturas siguen siendo muy simples, de 2 y 3 capas convolucionales cada una. Usando las técnicas de ajuste lineal la mejor arquitectura es la de tres capas 3C con función de pérdida MAE. Cuando usamos la discretización y posterior clasificación la mejor arquitectura es la de dos capas 2C con función de pérdida MAE. Es importante destacar que la de tres capas 3C con menos filtros es mucho más eficiente en términos de tiempo de entrenamiento.

Finalmente también se realizó un AEC con aprendizaje supervisado con la tarea

de enderezar la imagen rotada, o sea, de recrear la imagen pero en la orientación correcta al decodificarla. De esta forma la idea a futuro es poder examinar el espacio latente que se genera en dicha red a través de redes complejas como las que utilizan las Dras. Carolina Tauro y Carolina Daza al pasarle como condiciones iniciales las codificaciones realizadas a la salida del encoder en vez de las condiciones aleatorias con que ellas utilizaron en sus trabajos. El objetivo es ver si es posible recuperar los patrones espacio-temporales que se observan en la corteza visual de los mamíferos a partir de las entradas al espacio latente (que en este modelaría la corteza visual) y así entender cómo se codifica la orientación en la región V1.

Un detalle de este trabajo es que el conjunto de datos utilizado es una buena elección para el entrenamiento de la red si lo que deseamos es evaluar rápidamente nuestras redes propuestas. Dado que el tamaño de las imágenes es chico y su contenido es relativamente simple es posible utilizar redes no muy complejas, sin tanta profundidad, pues agregar más capas ocultas conlleva la necesidad de acceder a mejores recursos informático e invertir más tiempo en entrenar las redes. Pero una vez corroborada la capacidad de reconocer ángulos de rotación esperamos poder diseñar nuevas arquitecturas más complejas, como lo son las mencionadas al inicio de este capitulo, que sean capaces de reconocer rotaciones en imágenes de mayor dimensionalidad (por ejemplo, en colores) y con mayor variabilidad de contenido, como pueden ser los conjuntos de datos COCO, CIFAR10 o Google Street View.

Bibliografía

Berzal, F. (2018). *Redes neuronales y deep learning*.

Daza Caro, Y. C. (2017). *Sincronización en redes neuronales con interacciones complejas*. PhD thesis, Universidad Nacional de Córdoba.

Domingos, P. (2015). *The master algorithm: How the quest for the ultimate learning machine will remake our world*. Basic Books.

Downing, K. L. (2015). *Intelligence emerging: adaptivity and search in evolving neural systems*. MIT Press.

Gleiser, P. M., Tamarit, F. A., et al. (2018). Interplay of network structure and dynamics in functional organization of the visual cortex. *Physical Review E*, 98(6):062307.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Grandini, M., Bagli, E., and Visani, G. (2020). Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*.

Hebb, D. (1949). *The Organization of Behavior: A Neuropsychological Theory*. Taylor & Francis.

Hertz, J. A., Krogh, A. S., and Palmer, R. G. (1991). Introduction to the theory of neural computation.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA.

Rich, E. (1985). Artificial intelligence and the humanities. *Computers and the Humanities*, 19(2):117–122.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

Sáez Trigueros, D. (2017). Correcting image orientation using convolutional neural networks - a hands-on introduction to deep learning applications.

Tauro, C. B. (2012). *Estudio de la formación de mapas visuales mediante el uso de modelos neuronales bidimensionales*. PhD thesis, FaMAF - Universidad Nacional de Córdoba.

Tauro, C. B., Tamarit, F. A., and Gleiser, P. M. (2012). Synchronization in lattice-embedded scale-free networks. *Physica A: Statistical Mechanics and its Applications*, 391(3):834–842.

Tauro, C. B., Tamarit, F. A., Gleiser, P. M., et al. (2014). Modeling spatial patterns in the visual cortex. *Physical Review E*, 90(4):042818.